National Library of Canada
Acquisitions and Bibliographic Services Branch
395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada
Direction des acquisitions et des services bibliographiques
395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

UNIVERSITY OF ALBERTA

SPREAD SPECTRUM MULTIPLE ACCESS WITH ORTHOGONAL CONVOLUTIONAL CODES FOR INDOOR DIGITAL RADIO

by

Richard Tsz Shiu Tse

©

A Thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Science.

DEPARTMENT OF ELECTRICAL ENGINEERING

Edmonton, Alberta

Fall 1992

National Library of Canada
Bibliothèque nationale du Canada

Canadian Theses Service
Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-77114-3

Canada

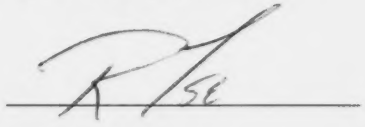
UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR:	**Richard Tsz Shiu Tse**
TITLE OF THESIS:	**Spread Spectrum Multiple Access with Orthogonal Convolutional Codes for Indoor Digital Radio**
DEGREE:	**Master of Science**
YEAR THIS DEGREE GRANTED:	**1992**

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as hereinbefore provided neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.



10759 University Ave.
Edmonton, Alberta
Canada, T6E 4P8

DATE: Oct. 9, 1992

The more abstract the truth you want
to teach the more you must seduce
the senses to it.

Friedrich Nietzsche,
Beyond Good and Evil

Wun, Too, Fweee!!

One of many math
teaching assistants

This time, everything is alright....
This time, everything is easy...

Bryan Adams

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **SPREAD SPECTRUM MULTIPLE ACCESS WITH ORTHOGONAL CONVOLUTIONAL CODES FOR INDOOR DIGITAL RADIO** submitted by **Richard Tsz Shiu Tse** in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE**.

W.A. Krzymien

W.D. Grover

P. Rudnicki

DATE: 5 Oct. 1992

Abstract

The use of very low rate orthogonal convolutional codes in an indoor digital radio direct sequence spread spectrum multiple access (DS-SSMA) system is investigated in this thesis. Computer simulations are performed to determine the average bit error rates obtainable by such a system in a soft partitioned office environment under different levels of multi-user interference. Three different methods are used to evaluate the performance of coded and uncoded SSMA systems: a complete system simulation for bit error rates greater than 10^{-4}, a simulation which counts encoded symbol errors and then uses an error bound calculation for bit error rates less than 10^{-4}, and a channel analysis method which predicts bit error rates for uncoded SSMA systems by examining the channel profiles. In the system, the spectral spreading of the data bits is accomplished by both the convolutional encoder and by partial Gold sequences of length proportional to the code rate. The interference level is determined by the number of interferers and the mean normalized cross-correlation between the partial Gold sequences. It is modelled as white Gaussian noise. The channel profiles used are generated by the SIRCIM software package, which generates channels with the same statistics as experimentally measured ones. This thesis shows that the use of some very low rate orthogonal convolutional codes can substantially increase the system capacity over an uncoded SSMA system when the bit error rate is set at acceptable levels for voice transmission.

Acknowledgements

I would like to thank Dr. Witold A. Krzymien for introducing me to this project and his supervision throughout the course of this work. I would also like to acknowledge Mike MacGregor, Jeff Kocuipchyk, and Bob Gregorish for supporting the computing system at Telecommunications Research Laboratories (TRLabs) while I was doing this work and Rohit Sharma for helping me with BOSS. A special thanks to all my fellow students at TRLabs, especially Brad Venables and Yvonne den Otter, for making my two years here so much fun.

The financial support offered by the Natural Sciences and Engineering Research Council of Canada (NSERC), the University of Alberta, and TRLabs is much appreciated.

Most of all, I would like to thank my parents, Sandy, and my grandmother for the incredible support they have given me throughout my life.

TABLE OF CONTENTS

1. Introduction 1

1.1 Thesis Overview 2

2. Background 4

2.1 Spread Spectrum Multiple Access 4

2.1.1 Principles of Spread Spectrum 4

2.1.2 Frequency Hopped SSMA 5

2.1.3 Direct Sequence SSMA 6

2.1.4 Advantages of SSMA 8

2.1.5 Disadvantages of SSMA 9

2.1.6 Gold Sequences for SSMA 10

2.2 Orthogonal Convolutional Codes 12

2.2.1 Orthogonal Convolutional Encoding 13

2.2.2 Decoding 15

2.2.3 Error Bounds in AWGN and Binary Symmetric Channels 20

2.3 Fading Multipath Channels 23

2.3.1 Flat Fading 24

2.3.2 Time Spreading 26

2.3.3 Frequency Spreading 27

2.4 Previous Work 28

3. Simulation Tools Used 30

3.1 Simulation of Indoor Radio Channel Impulse response Models (SIRCIM) 30

3.2 Block Oriented Systems Simulator (BOSS) 35

3.3 Fortran and C Support Programs 37

4. Overview of the SSMA System and Its Simulation Strategy 39
4.1 SSMA System 39
4.1.1 System Description 39
4.1.2 Full Simulation Strategy 47
4.1.3 BER Estimates Using Error Bounds and Channel Analysis 47
4.2 Interference Modelling System 51
5. Experiments and Results 57
5.1 Interference Modelling 57
5.1.1 Statistical Characterization of Interference 57
5.1.2 Cross-Correlation of Partial and Complete Gold Sequences 61
5.2 System Performance 64
5.2.1 BER Estimates Using Full Simulation 64
5.2.2 BER Estimates Using Error Bounds and Channel Analysis 68
5.3 Discussion of Results 75
6. Conclusions 80
6.1 Multi-user Interference Modelling 80
6.2 Simulation Methodology 80
6.3 Indoor Radio SSMA System Performance 81
6.4 Future Work 82
Bibliography 84
Appendix 1 Simulation Systems and Modules 87
Appendix 2 Fortran and C Support Code 201

List of Tables

Table 1. Trellis states and their corresponding codewords for an orthogonal convolutional code with constraint length K = 3. 14

Table 2. Memory requirements of Viterbi decoders for different code rates. 20

Table 3. Results of Kolmogorov-Smirnov goodness-of-fit test for multiple access interference and Gaussian distribution. 59

Table 4. Mean and maximum cross-correlations between partial and complete Gold sequences. 62

Table 5. Comparison of simulation symbol error rate, bit error rate, and bit error rate bound for SSMA system using 1/32 code with 30 users. 72

Table 6. Comparison of simulation bit error rates and bit error rate bounds for some SSMA systems. 73

Table 7. Average BERs for different SSMA systems determined with the error bound method. 74

Table 8. Average BER values from the channel analysis method and full simulation method for the uncoded SSMA system. 75

List of Figures

Figure 1. Power spectral densities of a) original signal, b) spread signal, c) spread interference signal, and d) despread desired signal and interference.....5

Figure 2. Diagram of operation of direct sequence spreading and despreading.....6

Figure 3. Diagram of direct sequence despreading of interference signal. 7

Figure 4. Outputs of integrate and dump filtering on Despread Output waveforms from a) Figure 2 and b) Figure 3. 8

Figure 5. Example of spreading with a) complete PN sequences and b) partial PN sequences. 12

Figure 6. Diagram of an orthogonal convolutional encoder. 13

Figure 7. Trellis diagram for an orthogonal convolutional code with constraint length K = 3.....14

Figure 8. Diagram of Viberbi algorithm selection of surviving paths. 16

Figure 9. Block diagram of an orthogonal convolutional decoder for code of constraint length K = 3.....18

Figure 10. Long-term path loss along three hypothetical paths, their mean path loss, and free space path loss. 25

Figure 11. The short-term fading of r(t) superimposed on its long-term fading pattern.....25

Figure 12. Intersymbol interference caused by time delay spread due to multipath propagation. 27

Figure 13. Diagram of a set of soft partitioned office multipath channels. 35

Figure 14. Diagram of a BOSS system.....36

Figure 15. Block diagram of the analyzed SSMA system. 39
Figure 16. A RAKE demodulator for DPSK signals. 43
Figure 17. BOSS implementation of the SSMA system. 46
Figure 18. Frequency spectrum of interference before despreading. 59
Figure 19. Frequency spectrum of interference after despreading. 60
Figure 20. Graph of normalized mean cross-correlations between partial and complete Gold sequences. 63
Figure 21. Performance of the SSMA system with no FEC coding. 66
Figure 22. Performance of the SSMA system with code rate 1/8. 66
Figure 23. Performance of the SSMA system with code rate 1/32. 67
Figure 24. Performance of the SSMA system with code rate 1/128. 67
Figure 25. Performance of the SSMA system with code rate of 1/1024. 68
Figure 26. Graph of data bit error probability bound versus encoded symbol error probability for orthogonal convolutional codes of constraint length K = 7. 69
Figure 27. Graph of data bit error probability bound versus encoded symbol error probability for orthogonal convolutional codes of constraint length K = 3. 69
Figure 28. Graph of data bit error probability bound versus encoded symbol error probability for orthogonal convolutional codes of constraint length K = 5. 70
Figure 29. Graph of data bit error probability bound versus encoded symbol error probability for orthogonal convolutional codes of constraint length K = 10. 70
Figure 30. Summary of performance of the SSMA system using different code rates. 76

List of Symbols and Abbreviations

Symbols

A_k - the average path gain at discrete time T_k
B_c - coherence bandwidth
$\delta(t)$ - impulse function
Δ_{ds} - time delay spread of channel
E_b - energy per data bit
E_s - energy per symbol
f_d - Doppler shift
K - the constraint length of a convolutional code
λ - wavelength of the carrier
N_c - the length of the partial PN sequence
N_o - one-sided power spectral density
p - probability of an encoded symbol error
P_b - data bit error probability
r - the code rate
SIR - signal to interference ratio
t_c - coherence time
T - symbol time
T_c - chip time
T_{imp} - time between consecutive channel sample points
T_k - discrete time delay for the k^{th} channel sample point
T_{seq} - duration of one complete PN sequence
xcorr - the normalized mean of the partial sequence cross-correlation magnitude
Z - generic parameter used in error bounding

Abbreviations

ACS - add-compare-select
AWGN - additive white Gaussian noise
BER - bit error rate
BOSS - Block Oriented Systems Simulator
BPSK - binary phase shift keying
BSC - binary symmetric channel
DPSK - differential phase shift keying
DS-SSMA - direct sequence spread spectrum multiple access
FDMA - frequency division multiple access

FEC - forward error correction
FH-SSMA - frequency hopped spread spectrum multiple access
ISI - intersymbol interference
LOS - line-of-sight
LPI - low probability of interception
OBS - obstructed
PN - pseudo-noise
SSMA - spread spectrum multiple access
SIRCIM - Simulation of Indoor Radio Channel Impulse response Models
TDMA - time division multiple access
T-R - transmitter to receiver
X-OR - the logical operation of exclusive OR

1. Introduction

The concept of radio telephony is appealing because it gives the user the freedom of mobility while still allowing him to receive important messages and information. Because this concept is so appealing, the field of mobile communications has exploded in recent years. This growth has created a problem with traffic congestion over the allocated spectrum for wireless phones and other wireless communication devices.

Also, indoor digital radio systems have to operate in a hostile environment of multipath fading radio channels. Consequently, they must be able to withstand variations of the channel parameters, occasional very low signal-to-noise ratios, and substantial intersymbol interference caused by frequency selective fading. The technique of direct sequence spread spectrum multiple access (DS-SSMA) has the inherent ability to deal with these problems. DS-SSMA is believed to be able to provide a higher traffic capacity than the more conventional techniques of frequency division multiple access (FDMA) and time division multiple access (TDMA) [1]. At the same time, certain features of DS-SSMA make it easier to implement; e.g. it may not require precise synchronization of mobile transmission bursts which is essential in TDMA systems. The use of forward error correction (FEC) codes in a DS-SSMA system can further improve its performance [2].

The performance of SSMA systems using low rate convolutional codes and operating in an AWGN channel was considered in [3] and [4]. The convolutional codes provided both error control and at least partial spectral spreading, and were shown to improve performance over the corresponding uncoded SSMA system for a given complexity, chip rate, and throughput. In fact, the very low rate orthogonal convolutional codes investigated in [4] allowed

the aggregate data rate of all simultaneous users to approach the Shannon channel capacity in the presence of background Gaussian noise when coordinated processing was implemented.

This thesis investigates the use of very low rate orthogonal convolutional codes in a non-coherent indoor digital radio DS-SSMA system. The system operates under channel conditions found in soft partitioned offices with no line-of-sight between the transmitter and receiver. The investigation is done by performing computer simulations of such a system. The DS-SSMA system is simulated using the Block Oriented Systems Simulator (BOSS), a commercially available simulation package [5]. The channel profiles are generated using the Simulation of Indoor Radio Channel Impulse response Models (SIRCIM) [6,7] software package. The system is evaluated in terms of the bit error rate (BER) for different codes and at different levels of multi-user interference.

For typical voice applications, BERs in the range 3×10^{-2} [8] to 1×10^{-3} are deemed acceptable. For 32 kbits/s packet speech applications using 1000 bit packets and speech interpolation for word error rates of 1×10^{-1}, bit error rates around 1×10^{-4} are required [9]. The maximum number of simultaneous users that can be accommodated at BER $\leq 3\times10^{-2}$, BER $\leq 1\times10^{-3}$, and BER $\leq 1\times10^{-4}$ are determined for systems using different code rates.

1.1 Thesis Overview

This thesis is divided into 6 chapters and two appendices which discuss background information and the work performed for and resulting from this thesis.

Chapter 2 contains the background material which is necessary for the reader to understand all the concepts discussed in this thesis. This material includes the concept of spread spectrum multiple access (SSMA), orthogonal

convolutional encoding, and fading multipath channel propagation. An overview of previous work done in the area of spread spectrum multiple access and personal radio communications is also included in this chapter.

Chapter 3 contains a discussion on the tools used to simulate the system. The software packages SIRCIM and BOSS are reviewed.

Chapter 4 describes the system models built and the simulation strategy used.

Chapter 5 presents and discusses the results of the computer simulations.

Chapter 6 presents the conclusions that can be drawn from the thesis work.

Details of the simulation modules and other support programs used are given in Appendix 1 and Appendix 2 respectively.

2. Background

An understanding of three topics is important to understanding the work to be presented. These topics are: spread spectrum multiple access, orthogonal convolutional codes, and fading multipath channels.

2.1 Spread Spectrum Multiple Access

Spread spectrum techniques have long been used in military communications because of their low probability of interception (LPI) due to their low energy density, and good jamming resistance. Recently, commercial applications for spread spectrum have exploded because of spread spectrum's potentially efficient spectrum use. It is especially promising for multiple access networks in today's congested radio spectrum: capacity of cellular telephone systems can theoretically be increased by a factor of 40 when compared to analog Frequency Division Multiple Access (FDMA) and by a factor of 4 when compared to digital Time Division Multiple Access (TDMA) [10]. Also, spread spectrum techniques can be helpful in combatting the problem of multipath signal propagation which occurs in mobile wireless communications [11].

2.1.1 Principles of Spread Spectrum

There are two principle types of spread spectrum systems, direct sequence (DS-SSMA) and frequency hopping (FH-SSMA). Both techniques take the original signal and spread its energy over a much wider bandwidth before transmitting, thus reducing the power per unit bandwidth per transmitter. The ratio of the spread bandwidth to the original signal bandwidth is known as the processing gain of the system. To properly receive the signal, its energy must be despread by a coordinated receiver (one with the identical spreading code as the transmitter). The need for the receiver to have the same spreading

code is the premise for multiple access using spread spectrum techniques: ideally, the interference at the receiver does not get despread, and thus, its power in the despread signal band remains low when compared to the despread desired signal as shown in Figure 1.

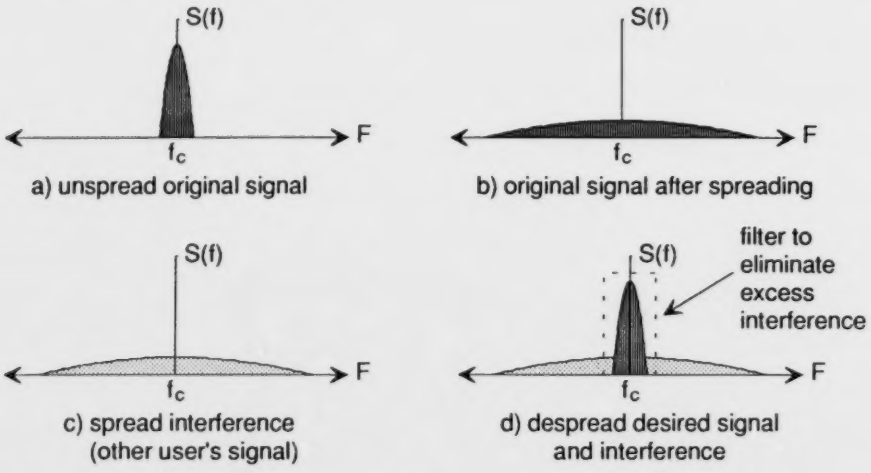


Figure 1. Power spectral densities of a) original signal, b) spread signal, c) spread interference signal, and d) despread desired signal and interference.

2.1.2 Frequency Hopped SSMA

FH-SSMA spreads its signal by hopping (varying) the carrier frequency of its transmission. This hopping can occur many times per bit (fast FH-SSMA), once per bit, or once every several bits (slow FH-SSMA). Interference results from other users' signals hopping onto the same carrier frequency as the desired signal. The pattern in which the frequencies are hopped is determined by a pseudo-noise (PN) sequence. A high capacity FH-SSMA system requires many filters tuned to specific hopping frequencies on the receiving end and a frequency synthesizer capable of abrupt and perhaps high speed frequency hops at the transmitting end. This system could be costly and therefore less suitable for high volume consumer products such as personal wireless

communication equipment. Therefore, only DS-SSMA is considered in this study.

2.1.3 Direct Sequence SSMA

DS-SSMA accomplishes spreading by multiplying the binary input by a unique binary PN sequence of a much higher rate. If the receiver multiplies its received signal with an identical binary PN sequence which is synchronized, the signal will be despread and upon signal detection with an integrate and dump filter, the original binary input will be recovered. This process is shown in Figure 2: if the Data stream is multiplied with the PN Sequence stream, the result is the Spread Output stream. Multiplying the Spread Output stream with the synchronized PN Sequence stream again will result in the Despread Output stream which is identical to the Data stream.

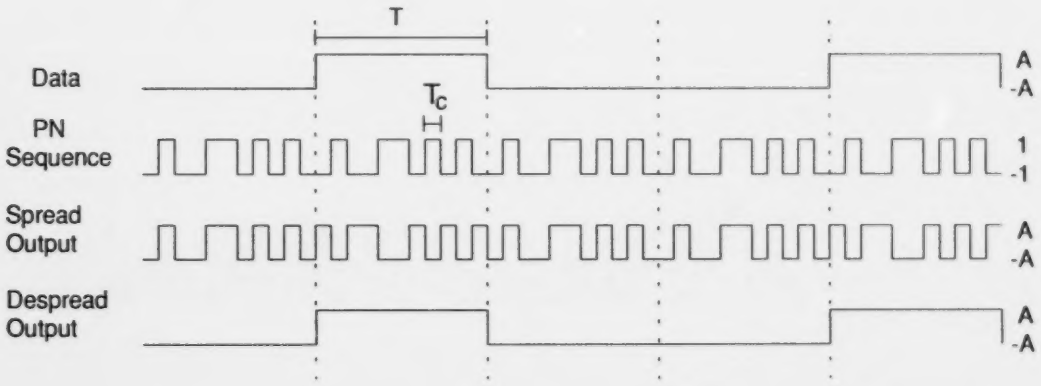


Figure 2. Diagram of operation cf direct sequence spreading and despreading.

Interfering signals are not despread if the cross-correlation between the interfering signal's PN sequence and the receiver's PN sequence is very low when normalized to the sequence's length. The lower this value, the lower the interference level as the integrate and dump filter at the receiver will return a

near-zero value. In Figure 3, an interfering signal which uses a PN sequence with a normalized cross-correlation equal to T_c/T (ie. the processing gain which, in this case, is equal to 1/11) is despread. T_c is the chip time.

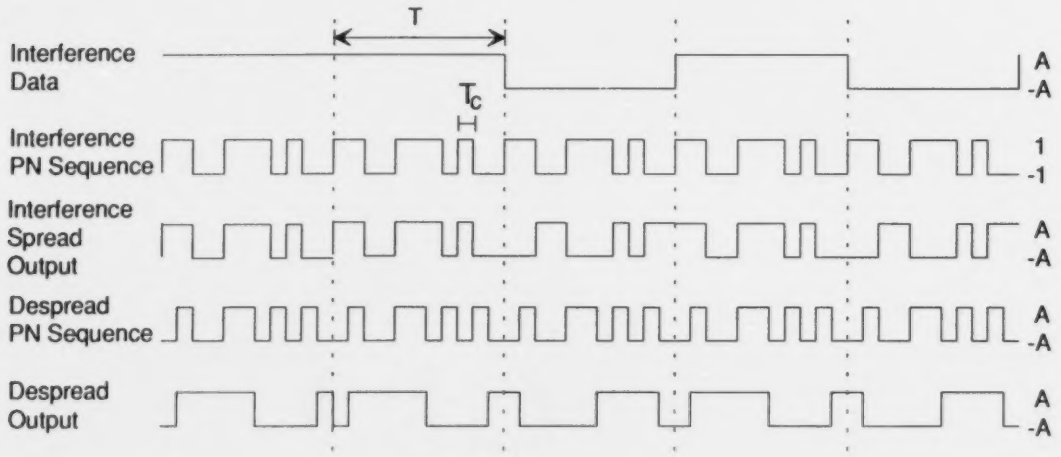


Figure 3. Diagram of direct sequence despreading of interference signal.

Performing the integrate-and-dump operation over periods of T on the Despread Output waveforms from Figure 2 and Figure 3 would give detected signal energies of A^2T for the desired user and $A^2T/11$ for the interfering user as shown in Figure 4.

The use of sequences with low normalized cross-correlations (and thus high processing gains) is the key to building a high capacity SSMA system. Gold sequences are such a set of codes. They will be discussed in Subsection 2.1.6. The capacity of an SSMA system is thus limited by the interference produced by other users allocated to the same bandwidth.

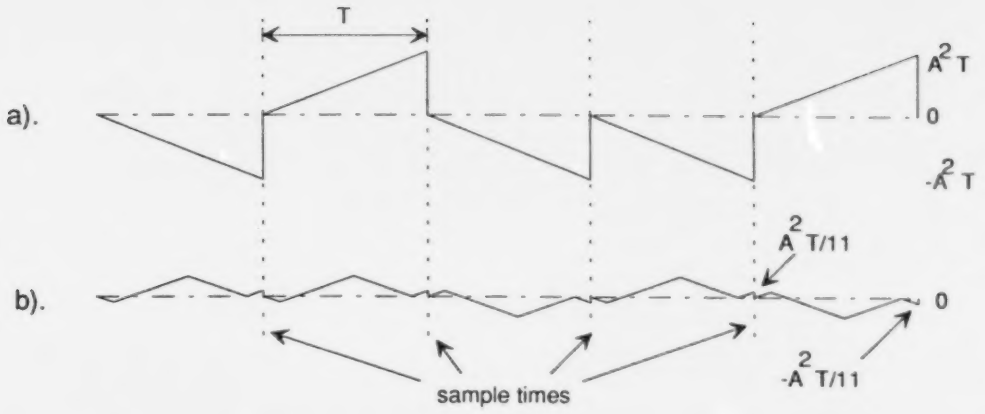


Figure 4. Outputs of integrate and dump filtering on Despread Output waveforms from a) Figure 2 and b) Figure 3.

2.1.4 Advantages of SSMA

The first advantage of SSMA is graceful degradation. As more users transmit signals, the interference increases, and everyone's signal quality degrades. Thus, the practical limit on the maximum number of users is soft and is determined by the degradation of signal quality that each user can tolerate. The only hard limit on the number of users in the system is determined by the number of PN sequences available. However, in a practical system, this limit cannot be reached because the signal quality would be intolerable well before this point. In FDMA and TDMA, the number of channels available determines the maximum number of users.

The second advantage is increased capacity via user activation factor. In FDMA and TDMA, each user occupies one channel for the entire time that he is on the system, even if he is not transmitting any data. In SSMA, if a user is on the system but not actively transmitting data, his signal can be shut off until he has information to transmit again. This reduces interference to the other users. In voice calls, a person usually talks for 40-50% of the time [12]. This would allow the capacity of a SSMA system to approximately double.

Capacity can be further increased via antenna sectorization. Use of highly directional transmitting and receiving antennas at the fixed antenna locations used by the base stations can decrease the level of interference seen by both the mobile users and the base station.

Another advantage of using SSMA is its ability to resolve and then suitably combine the signal energy coming through different paths when the signal propagates through a multipath channel. This can be accomplished using a RAKE receiver. The operation of this receiver is discussed in Section 4.1 and [13].

Asynchronous multiple access is another advantage offered by SSMA. Precise synchronization of mobile transmissions bursts such as that required by TDMA may not be necessary. This makes SSMA systems easier to implement.

2.1.5 Disadvantages of SSMA

DS-SSMA systems suffer from one major drawback: the need for power control for the reverse link (mobile to base station) to combat the near/far problem. Without reverse link power control, an interfering user (mobile transmitter) that is much closer to the receiver (base station) than the desired user would transmit a signal that would swamp the desired one. Power control for the forward link (base station to mobile) is not required because all signals originating at the base station will have the same power at the mobile.

If in the example given in Subsection 2.1.3 the interfering user's signal was at an amplitude of 11A instead of A, then the energy of the interference after the integrate and dump filter would be $(11A)^2T/11 = 11A^2T$ which is greater than the energy of the desired signal. Much research is being done on practical systems to accomplish power control, and commercially effective systems are on the verge of being released [14].

2.1.6 Gold Sequences for SSMA

Because the cross-correlation between spreading sequences of different users directly affects the interference levels generated in a multiple access system, sequences with low normalized cross-correlations values must be used to maximize system capacity. Gold sequences [15] are such a set of sequences. In this study, the Gold sequences were modified by adding an additional chip to the end of the sequence. This was done to make the sequence length a power of two. Since the codewords of the orthogonal convolutional codes that are used in this study also have lengths which are a power of two, the modification allows the starting points of data bits and Gold spreading sequences to remain aligned. While this adversely affects the normalized cross-correlation properties of the Gold sequences, it simplifies the hardware required to find the starting point of each codeword.

Original Gold sequences are generated by combining two maximal length sequences. 2^n+1 codes can be generated with two maximal length sequences of length 2^n-1. The two maximal length sequences must be carefully chosen so that the resulting Gold sequences exhibit the cross-correlation properties desired. A method for doing this is described in Appendix 7 of [15] using the irreducible polynomial tables from [16].

The Gold sequences have auto and cross-correlations which are bounded by [13]:

$$|A_{corr}| = 2^n - 1 \qquad (2.1.6.1)$$

$$|X_{corr}| \le 2^{\frac{n+1}{2}} + 1 \quad \text{for n odd} \qquad (2.1.6.2)$$

$$\le 2^{\frac{n+2}{2}} + 1 \quad \text{for n even}$$

Thus, the gain of the desired signal over an interfering signal in a system utilizing these sequences is:

$$|A_{corr}|/|X_{corr}| \geq \frac{2^n - 1}{2^{\frac{n+1}{2}} + 1} \quad \text{for n odd} \qquad (2.1.6.3)$$

$$\geq \frac{2^n - 1}{2^{\frac{n+2}{2}} + 1} \quad \text{for n even}$$

From this, we can see that for large n, the gain approximately doubles whenever n is increased by 2.

For the modified Gold sequences used in this study, the autocorrelation value would be 2^n and the maximum and mean cross-correlations would increase by small factors which can be determined experimentally.

When one Gold sequence is used to spread more than one symbol in a multiple access system, then the cross-correlation of partial Gold sequences must be investigated. The mean of these cross-correlations will be higher than the mean of the cross-correlations between complete Gold sequences of shorter length equal to the length of the partial sequences. However, since the number of Gold sequences available is approximately equal to the length of the sequence, short Gold sequences severely limit the maximum number of users when very low code rates are used in an SSMA system with a fixed bandwidth (such as the one being investigated in this thesis). Thus, longer sequences must be used and the spreading of the encoded symbols is done with partial sequences.

The procedure of spreading with complete and partial PN sequences is shown below. In Figures 5a and 5b, the chip rates are identical, but the length of the PN sequence of Figure 5b is double that of Figure 5a. Results of an investigation into the mean cross-correlations of complete and partial modified Gold sequences are given in Subsection 5.1.2.

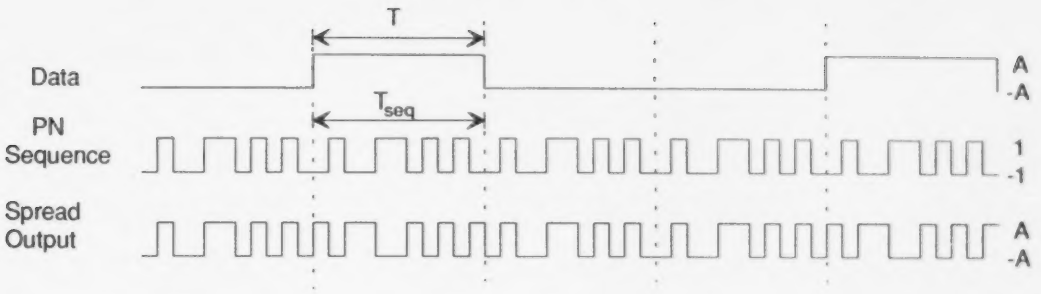


Figure 5a

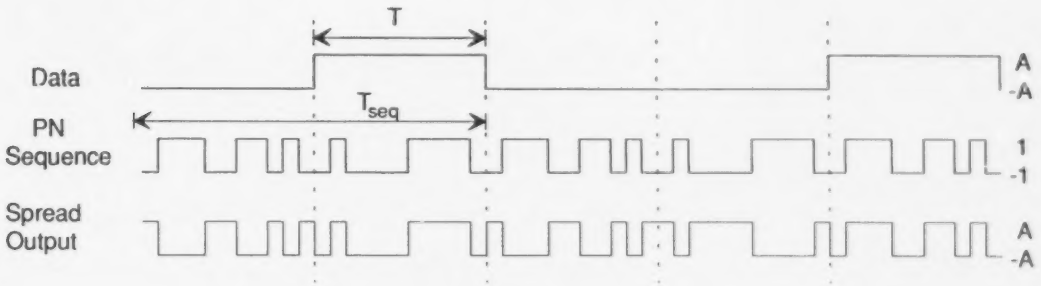


Figure 5b

T_{seq} is the duration of one PN sequence

Figure 5. Example of spreading with a) complete PN sequences and b) partial PN sequences.

2.2 Orthogonal Convolutional Codes

Convolutional codes are FEC codes which are generated by passing the binary data through a finite state shift register and then manipulating the bits in the shift register (usually by X-ORing some of them) to generate codewords [17]. This encoding structure introduces correlation among transmitted symbols which can be exploited to improve the receiver's performance. That is, knowing the encoder structure, previous transmitted codewords, and the current codeword, a better estimate of the actual transmitted signal can be made than if the codewords were treated as independent block codes. Orthogonal convolutional codes are a subset of convolutional codes. They are especially useful if very low

code rates are desired because their structure allows very simple encoder and decoder designs. This allows the use of very powerful codes without requiring very complex hardware.

2.2.1 Orthogonal Convolutional Encoding

In the orthogonal convolutional codes used in this thesis, the output codewords have the same structure as the codewords of Hadamard codes. They are generated by taking a convolutional code register (a shift register of length K which equals the code's constraint length) and passing the shift register's output bits through an Hadamard block coder as shown in Figure 6 [4]. The upper register is the convolutional code register and the lower register is the Hadamard block coder.

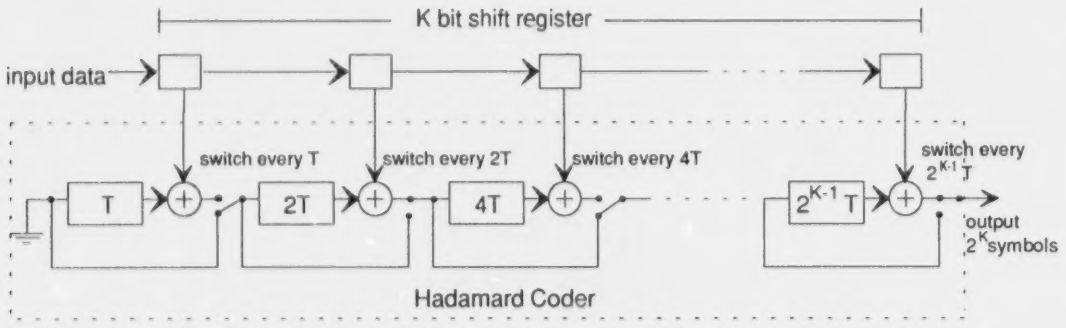


Figure 6. Diagram of an orthogonal convolutional encoder.

One codeword of length 2^K is generated for each input bit. This results in an FEC code of rate $r = 2^{-K}$ with a free distance of $d = 2^{K-1}$ [18]. Thus, very low rate codes can be generated with the very simple digital circuit shown above.

As for other convolutional codes, a trellis diagram can be drawn to show the possible paths that the coder can take. The trellis diagram for an orthogonal convolutional code with constraint length K = 3 is shown in Figure 7. The codewords and their corresponding trellis states are listed in Table 1.

Trellis State	Codeword
000	00000000
100	01010101
010	00110011
110	01100110
001	00001111
101	01011010
011	00111100
111	01101001

Table 1. Trellis states and their corresponding codewords for an orthogonal convolutional code with constraint length K = 3.

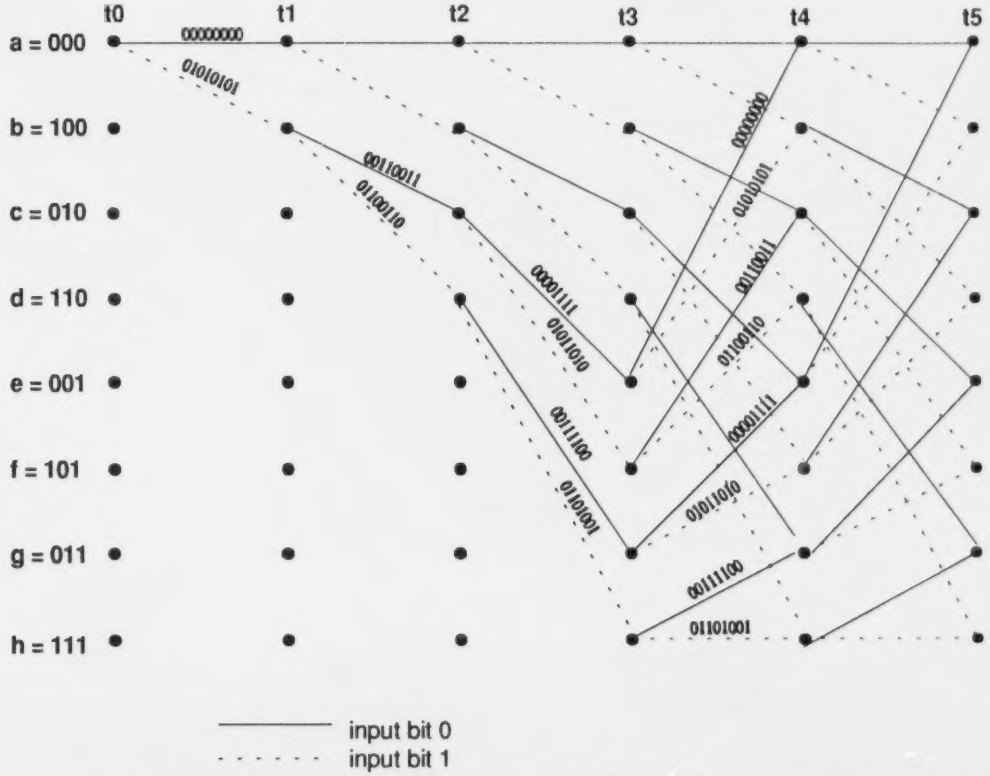


Figure 7. Trellis diagram for an orthogonal convolutional code with constraint length K = 3.

2.2.2 Decoding

The optimal decoder for any convolutional code is the Viterbi decoder. This decoder uses the trellis structure of the code and traces the maximum-likelihood path. To do this, the distances between the received codeword and each valid codeword in the trellis are calculated. The value is called a branch metric. Hard-decision Viterbi decoding is discussed here.

If on-off signalling is used, the codewords are as given in Table 1 and each branch metric equals the codeword length minus the Hamming distance between the received codeword and the valid branch codeword. If polar signalling is used, then 0 is replaced with +1, and +1 is replaced with -1 in the codewords from Table 1. Each branch metric is calculated as:

$$M = \sum_{n=1}^{2^K} y_n^{received} y_n^{valid} \qquad (2.2.2.1)$$

where $y_n^{received}$ is the n^{th} symbol of the received codeword and y_n^{valid} is the n^{th} symbol of the valid codeword corresponding to the branch. This branch metric is equivalent to the cross-correlation between the received codeword and the valid codeword in the trellis. This can also be expressed as the codeword length minus the Euclidean distance between the received codeword and the valid codeword in the trellis.

For a binary data signal, each trellis state has two possible input branches. For each trellis state at the latest stage of the trellis, the algorithm traces back through the branches which are connected and sums the branch metrics. The path which has the highest metric sum is the more likely path and is deemed the surviving path. At the completion of this step, each state at the latest stage of the trellis should have one incoming surviving path linked to it.

When all surviving paths merge to one past state, this state is the maximum-likelihood decoded state and the uncoded information can be derived from it.

This process is illustrated below for the same K=3 encoder discussed in Subsection 2.2.1 when the input data sequence, the transmitted codewords, and the received codewords are as given below. For simplicity, the decoder here operates on on-off signalled inputs. The branch metrics are given on the trellis decoding diagram. The surviving paths are highlighted in bold, and the final path metrics for each state in the latest trellis stage are given as M.

Input data sequence: 1, 0, 0, 1, 0 ...

The transmitted codewords are: 01010101, 00110011, 00001111, 01010101, 00110011 ...

The received codewords are: 01010101, 01100011, 00011111, 11011111, 00110011 ...

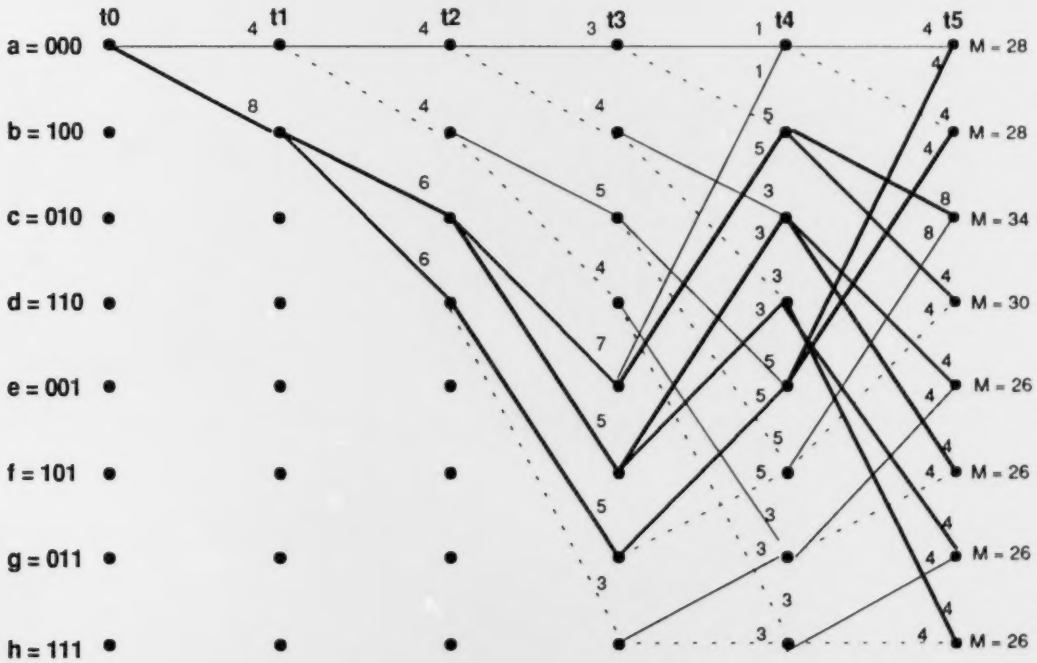


Figure 8. Diagram of Viterbi algorithm selection of surviving paths.

Only the first transmitted bit can be decoded at the instant t5 because at the later stages, all the surviving paths have not yet merged into one. In an ideal Viterbi decoder, the path metrics would be kept in memory until all surviving paths merged into one.

However, such a system cannot be implemented practically because it would require an indeterminate memory size to store all past branch metrics of the surviving paths until they all merged to one. In practical systems, the path memory is truncated. In other words, the decoder traces the trellis back a set number of branches and then is forced to make a decision at this point. It has been shown that if the path memory is set at 5 times the constraint length, and the best path metric is taken at the truncation point, the loss in coding gain is on the order of 0.1 dB [13,18]. This method also gives the advantage of a constant decoding delay.

For the orthogonal convolutional codes used in this thesis, a simple device to compute the branch metrics can be made which takes advantage of the special structure of the codewords. This device is known as the Green Machine [19]. The advantage of using the Green Machine is that the processing complexity of the decoder grows only linearly with the code constraint length (although the decoder memory still grows exponentially) since the branch metrics are calculated serially. There are exactly 2K additions and subtractions per symbol time and the speed of these calculations need only be equal to the encoded symbol rate. Figure 9 is a block diagram of a possible hardware design of a hard-decision orthogonal convolutional decoder for K = 3.

If the input symbols are, in chronological order: x1, x2, x3, x4, x5, x6, x7, and x8, where {x1, x2, ... x8} $\in$ {+1, -1}, then the output of the Green Machine serial decoder will, in chronological order, equal:

[(x1+x2)+(x3+x4)]+[(x5+x6)+(x7+x8)]

[(x1+x2)+(x3+x4)]-[(x5+x6)+(x7+x8)]

[(x1+x2)-(x3+x4)]+[(x5+x6)-(x7+x8)]

[(x1+x2)-(x3+x4)]-[(x5+x6)-(x7+x8)]

[(x1-x2)+(x3-x4)]+[(x5-x6)+(x7-x8)]

[(x1-x2)+(x3-x4)]-[(x5-x6)+(x7-x8)]

[(x1-x2)-(x3-x4)]+[(x5-x6)-(x7-x8)]

[(x1-x2)-(x3-x4)]-[(x5-x6)-(x7-x8)]

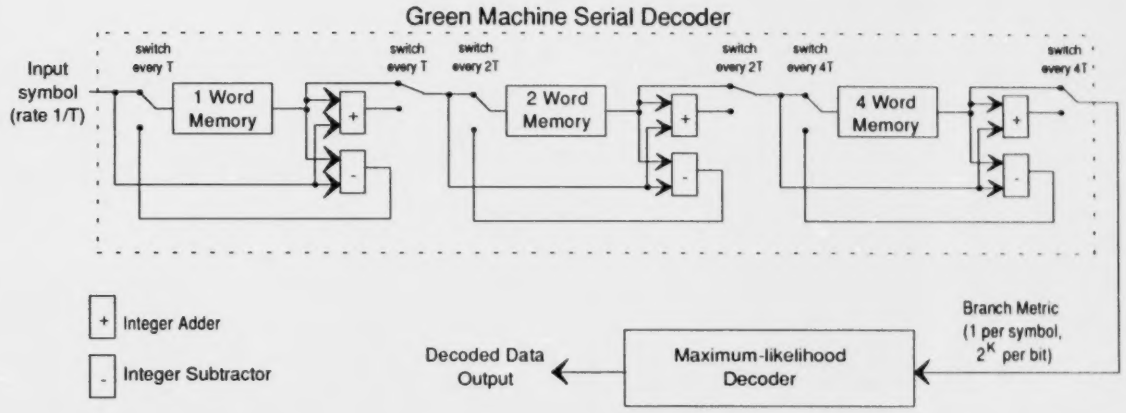


Figure 9. Block diagram of an orthogonal convolutional decoder for code of constraint length K = 3.

One output is generated every symbol period. If the codewords in Table 1 are converted to polar signals (0 → 1, 1 → -1) and fed through the serial decoder, then one of the above outputs will equal the maximum branch metric value of 8 while the rest will equal 0 for each codeword.

After the serial decoder, the branch metrics are fed to the maximum-likelihood decoder. This module adds the branch metrics to the appropriate paths and then determines the surviving path reaching each state. These

surviving paths must be stored in memory. The best of these surviving paths is then deemed to be the most likely path and is traced back to the decoder's truncation point to find the trellis state at that point. The most likely transmitted data bit can then be determined from this state. This add-compare-select (ACS) operation is standard to Viterbi decoders. The truncated path memory decoding operation is only optimal if all the surviving paths have merged prior to the truncation point. Otherwise, small losses in the coding gain are introduced.

The memory requirements of a Viterbi decoder increase exponentially with constraint length. A decoder for a code of constraint length K and with a path memory truncated to 5K branches would have the following memory requirements:

a). $2^K \times$ (Ceiling[ln($5 \times K \times 2^K$)/ln(2)] bit memory) to store all surviving path metric values at the latest stage of the trellis (maximum path metric value is $5 \times K \times 2^K$, thus requiring Ceiling[ln($5 \times K \times 2^K$)/ln(2)] bit memories, and there are 2^K paths). The Ceiling[x] function rounds x to the smallest integer larger than or equal to x.

b). ($5 \times K \times 2^K$)$\times$(K+1 bit memory) to store all the branch metrics and flags in the surviving paths (each branch has a maximum metric of 2^K, thus requiring K bits, and there are a maximum of $5 \times K \times 2^K$ branches in the surviving paths). The extra bit is for flagging whether its surviving path resulted from a +1 or a -1 data input).

d). a maximum of $(1+2+4+...2^{K-1}) \times$ (K-1 bit memories) for the serial decoder.

For a K=7 decoder, the memory requirements would be a) 128×13 bit memories, b) 4480×8 bit memories, and c) $(1+2+4+8+16+32+64) \times 6$ bit

memories. The total number of bits is 38266 which is just under 5 kbytes of memory. If standard 4-bit word memory is used, then 38904 bits, or around 5 kbytes of memory are required. Memory requirements using the 4-bit word memory chips for various code rates are shown in Table 2.

K	Memory Required (bytes)
uncoded	0
3	72
5	864
7	4863
10	80383

Table 2. Memory requirements of Viterbi decoders for different code rates.

2.2.3 Error Bounds in AWGN and Binary Symmetric Channels

The transfer function of an orthogonal convolutional code of rate 1/n is [18]:

$$T_K(D,I) = \frac{ID^{\frac{Kn}{2}}(1-D^{\frac{n}{2}})}{1-D^{\frac{n}{2}}[1+I(1-D^{\frac{n(K-1)}{2}})]} \qquad (2.2.3.1)$$

where in the expanded $T_K(D,I)$, the exponent of I denotes the number of bit errors resulting from the wrong path selection, the exponent of D denotes the path weight, and K is the constraint length of the code. The transfer function describes all possible paths that can be taken when one deviates from and then returns to the all zeros path in the trellis. The expanded $T_K(D,I)$ is thus a polynomial with an infinite number of terms, representing the infinite variety of paths that can be taken through the trellis before returning to the all zeros path. In the expanded $T_K(D,I)$, the coefficient of D in each term is the number of paths

of distance d_{path} (where d_{path} is the exponent of D) that can be traversed in the trellis of the code when the paths deviate from and then return to the all zeros path. Each term will also contain the variable I. The exponent of I in each term gives the number of bit errors which will occur as a result of following any of the paths represented by the term in the expanded $T_K(D,I)$. Therefore, this function can be used to upper bound the bit error rate of the code in an additive white Gaussian noise (AWGN) or binary symmetric channel.

The bit error probability bound can be found by taking the derivative of the transfer function given above with respect to I and evaluating it at I = 1. By doing this, each term in the expanded version of the function is weighted by the exponent of I, thus giving the first-event error probability if the path corresponding to the term is taken. The detailed expression for the bound is given in [18] as:

$$P_b < \frac{\partial T_K(D,I)}{\partial I}\Big|_{I=1,D=Z} = \frac{Z^{\frac{Kn}{2}}(1-Z^{\frac{n}{2}})^2}{(1-2Z^{\frac{n}{2}}+Z^{\frac{Kn}{2}})^2} < \frac{Z^{\frac{Kn}{2}}}{(1-2Z^{\frac{n}{2}})^2} \qquad (2.2.3.2)$$

where, for a rate 1/n code and the binary-input AWGN channel, $Z^n = e^{-nEs/No} = e^{-Eb/No}$. E_s is the energy per coded symbol, $nE_s = E_b$ is the energy per input bit, and N_o is the one-sided power spectral density of the AWGN in the channel. Consequently:

$$P_b < \frac{e^{\frac{-KE_b}{2N_o}}}{(1-2e^{\frac{-E_b}{2N_o}})^2} \qquad \text{where } \frac{E_b}{N_o} > 2\ln(2) \qquad (2.2.3.3)$$

Since $E_b/(N_o\ln(2)) = C_T/R_T$ [18], this bound can be presented as follows:

$$P_b < \frac{2^{\frac{-KC_T}{2R_T}}}{[1-2^{1-\frac{C_T}{2R_T}}]^2} \qquad \frac{R_T}{C_T} < \frac{1}{2} \qquad (2.2.3.4)$$

where R_T is the transmission rate, C_T is the capacity, and both are in nats per second (1 nats/s = ln(2) bits/s). As shown in [18], this bit error probability can be more tightly bound by employing the Gallager bound. The result is:

$$P_b < \frac{2^{\frac{-KE_c(R_T)}{R_T}}}{[1-2^{-\delta(R_T)}]^2} \qquad 0 \le R_T < \frac{C_T}{1-\varepsilon_p} \qquad (2.2.3.5)$$

where:

$$E_c(R_T) = \frac{C_T}{2} \qquad 0 \le R_T < \frac{C_T}{2} \qquad (2.2.3.6)$$

$$= C_T - R_T/(1-\varepsilon_p) \qquad \frac{C_T}{2} \le R_T \le C_T(1-\varepsilon_p)$$

and $\delta(R_T) > 0$ for $\varepsilon_p > 0$.

This bound on bit error probability can also be presented as follows [4]:

$$P_b < \frac{2^{\frac{-KE_c(r)}{r}}}{[1-2^{-\delta(r)}]^2} \qquad \text{for } 0 < r < c(1-\varepsilon_p) \qquad (2.2.3.7)$$

with:

$$E_c(r) = \frac{c}{2} \qquad \text{for } 0 < r < \frac{c}{2} \qquad (2.2.3.8)$$

$$= c - \frac{r}{1-\varepsilon_p} \qquad \text{for } \frac{c}{2} \le r < c(1-\varepsilon_p)$$

where r is the code rate ($r=2^{-K}$), $c = E_s/(N_o \ln(2))$, and $\delta(r) > 0$ for $\varepsilon_p > 0$.

As shown in [4], this bound also holds for very noisy channels in which $\varepsilon^2/\sigma^2 << 1$; ε^2 is the mean symbol energy and σ^2 is the channel noise variance. In a system with a reasonably efficient bit signal-to-noise ratio, the use of the very low code rate r yields a very low symbol signal-to-noise ratio ε^2/σ^2. Hence, in a very low rate coded system, a very noisy channel operation is typical. For such a very noisy case [4]:

$$c = \frac{\varepsilon^2}{2\sigma^2 \ln(2)} \qquad (2.2.3.9)$$

For the k^{th} user of an SSMA system, this value of c becomes [4]:

$$c_k = \frac{S_k / (N_o W)}{\ln(2)[1 + \sum_{j \neq k} S_j / (N_o W)]} \qquad (2.2.3.10)$$

where S_k is the k^{th} user's power, and W is the system's bandwidth.

If hard-decision decoding is used, then $Z = \sqrt{4p(1-p)}$ as for a binary symmetric channel (BSC) [18]. The variable p represents the probability of an error in detecting a coded symbol and can be calculated for operation in AWGN channels for different detection methods using well known error probability formulas. Hence, from (2.2.3.2), the following bit error probability bound results from hard decision decoding:

$$P_b < \frac{(\sqrt{4p(1-p)})^{\frac{Kn}{2}}}{(1 - 2(\sqrt{4p(1-p)})^{\frac{n}{2}})^2} \qquad (2.2.3.11)$$

Consequently, acceptable bit error rates can theoretically be obtained when operating with very low rate orthogonal convolutional codes over white Gaussian noise (or interference) channels.

2.3 Fading Multipath Channels

In personal mobile radio communications, a single line-of-sight path between the transmitting and receiving antennas rarely exists. Instead, the signal propagates through many different paths which include reflections from walls, ceilings, and other obstacles to get to the receiver. As a result, the received signal consists of many components, each with a different phase and of a different amplitude. As well, since all propagation paths are not of the same length, each signal will arrive with a certain time delay (time delay spread) with

respect to the shortest path. Movement of the mobile transceivers also has an effect on the channel. The three main effects of these channels on the transmitted signal are: a) flat fading, b) time spreading, and c) frequency spreading.

2.3.1 Flat Fading

Flat fading results from signal propagation through space. Flat fading can be divided into two categories: long-term fading, and short-term fading. Long-term fading, also known as non-selective shadowing [20], is the average loss in signal strength at any particular location. It is caused by relatively small-scale changes in topography as the mobile transceiver moves. Averaging this long-term fading over different paths determines a path loss exponent for the environment. This value represents the average exponent value which the signal attenuates by as the distance between the transmitter and receiver changes [20]. It is usually greater than that of the typical line-of-sight free space path which is 2 (free space line-of-sight signals attenuate proportionally to $1/d^2$ where d is the distance between the transmitter and the receiver). The path loss exponent of a multipath channel is typically between 3 and 5 for multi-floored partitioned office buildings [7]. This may be because energy is absorbed by the obstacles which the signal reflects off or because destructive interference occurs between signals which propagate through different paths. Complete signal loss can occur if the receiver travels through an area where the transmitted signal cannot reach or is severely attenuated (ie. a deep fade region).

Figure 10 shows some hypothetical long-term fading patterns of three paths, their averaged path loss, and the path loss of an ideal line-of-sight free space channel given as a ratio of the received signal power P_R to the transmitted signal power P_T.

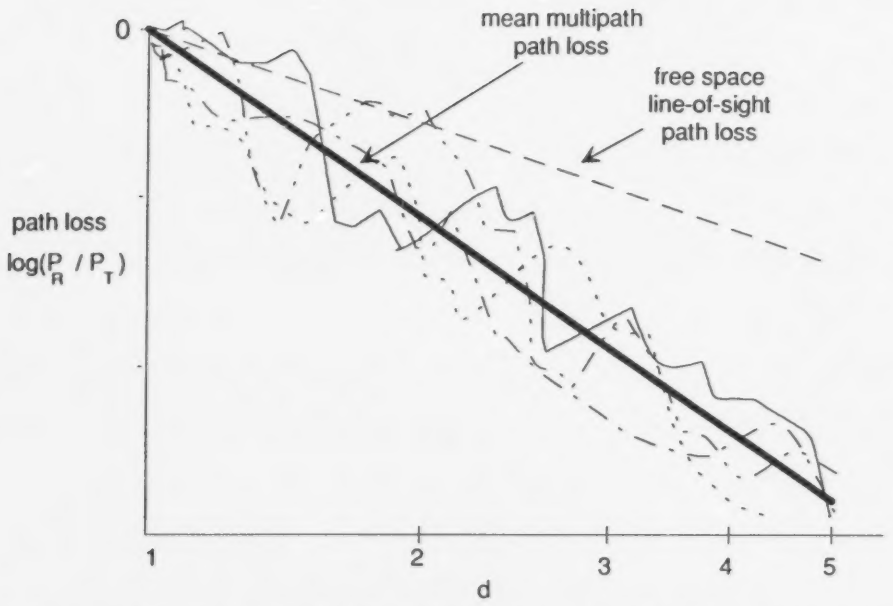


Figure 10. Long-term path loss along three hypothetical paths, their mean path loss, and free space path loss.

Short-term fading consists of the instantaneous, fast, and deep variations in signal strength at any particular location and time. It is also known as Rayleigh fading or envelope fading. Short-term fading is caused by multipath propagation and changes in the environment. Averaging the short-term fading over time will give the long-term fading value. Figure 11 shows the short-term fading of a signal r(t) superimposed over its long-term fading pattern.

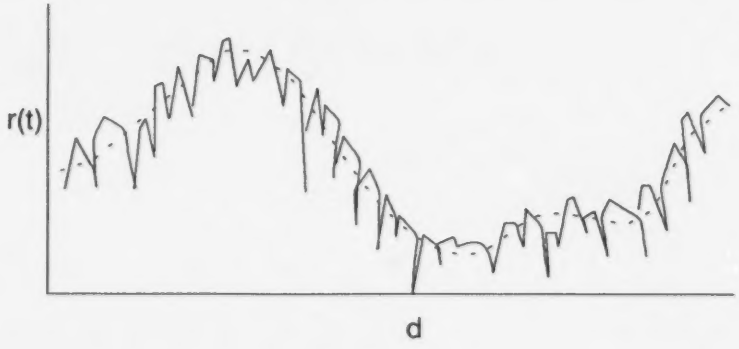


Figure 11. The short-term fading of r(t) superimposed on its long-term fading pattern.

2.3.2 Time Spreading

The multipath channel spreads the transmitted signal in time. This time delay spread Δ_{ds} is dependent on the channel. Typically, for indoor radio channels, Δ_{ds} is less than 500 ns [7]. An adverse effect of multipath fading on transmission integrity is increased intersymbol interference (ISI) due to the time delay spread. Severe ISI results if the time delay spread is greater than the symbol interval of the digital signal.

An example of intersymbol interference caused by time delay spread due to multipath propagation on a signal is shown in Figure 12. From the figure, it is obvious that the received signal is different from the transmitted binary sequence and that detection error can occur due to the intersymbol interference.

Coherence bandwidth is related to time dispersion. If we define the coherence bandwidth as the maximum bandwidth within which the phase correlation coefficient is to be greater than 0.5, then it can be estimated as [21]:

$$B_C = 1/(4\pi\Delta_{ds}) \qquad \text{for phase modulated systems} \qquad (2.3.2.1)$$

When the transmitted signal has a bandwidth greater than B_C, then two frequencies of the same signal will undergo uncorrelated attenuation and phase shifts. When this occurs, the channel is said to induce frequency selective or time dispersive fading.

For an indoor radio system with $\Delta_{ds} \cong 500$ ns, the coherence bandwidth is approximately equal to 160 kHz. A SSMA system will typically have a bandwidth of several MHz, thus time dispersion must be given serious consideration. Fortunately, spread spectrum has an inherent ability to take advantage of this time dispersion: a RAKE receiver may be used to resolve the different signal components. More information on the RAKE receiver is given in Section 4.1.

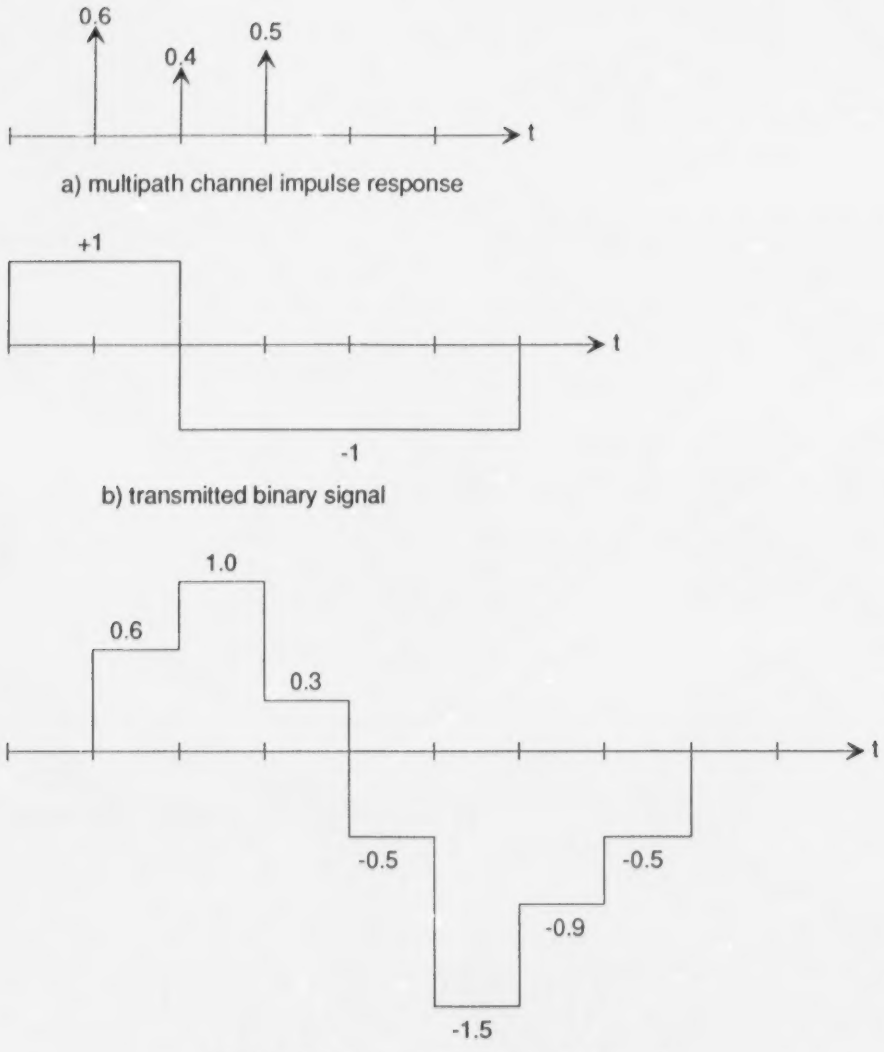


Figure 12. Intersymbol interference caused by time delay spread due to multipath propagation.

2.3.3 Frequency Spreading

A third multipath channel effect known as the Doppler effect results from the motion of the transmitter or receiver. If one is moving with respect to the other, an apparent frequency shift appears on the received carrier. This would make detection of digital signals based on phase or frequency modulation less

effective. The maximum frequency shift, or Doppler shift, can be calculated as [21]:

$$f_d = \max(V\cos\theta/\lambda) = V/\lambda \qquad (2.3.3.1)$$

where $V\cos\theta$ is the velocity of the receiver relative to the transmitter and λ is the wavelength of the transmitted signal.

When propagating through a multipath channel, the signal reaches the receiver from many different directions. Therefore, a frequency spread, or Doppler spread, appears on the received signal. The maximum Doppler spread is equal to twice the maximum frequency shift.

Coherence time is related to the Doppler spread. Identical signals which are transmitted with time separations equal to or greater than the coherence time will undergo uncorrelated attenuation and phase shifts. The coherence time can be calculated as follows [13]:

$$t_c = 1/(2f_d) \qquad (2.3.3.2)$$

When t_c is smaller than the duration of the transmitted symbol, then the channel induces time selective or frequency dispersive fading.

For a typical indoor radio system, $V\cos\theta$ can be approximately 1 m/s and λ can be around 0.22 m (for a 1.35 GHz carrier) which yields $f_d \cong 4.5$ Hz and $t_c \cong 0.11$ s . At reasonable bit rates and realistic transceiver velocities, the Doppler effect is negligible in indoor systems.

2.4 Previous Work

Previous work related to the area of spread spectrum multiple access in multipath channel environments is quite extensive. Most of it has shown that

SSMA techniques allow for reasonable system capacities when compared to FDMA or TDMA [1,22,23].

However, most of the papers published utilize spread spectrum in the conventional sense, where the chip rate is much higher than the data and the FEC encoded data rates [24,25,26]. However, it was shown in [3] that using lower rate convolutional codes with shorter PN spreading sequences (and thus incurring the penalty of greater normalized cross-correlation with interfering users) provided superior performance in an SSMA system. Very low code rates have not been widely investigated for SSMA systems except for [4], where the FEC spreading mechanism used in this thesis was introduced. In [27], the use of low rate convolutional codes in a spread spectrum communication system operating in an additive white Gaussian noise channel was analyzed. Both the analyses in [4] and [27] used coherent systems operating in a non-multipath channel environment.

Also, most papers utilized a Rayleigh or Rician model for the multipath channel characterization [24,25,26,28] where the multipath channel impulse response components were independent Rayleigh or Rician variables. The number of components of each multipath channel were also dictated by the particular investigation and not by the environment. The use of the SIRCIM channels will have an effect on the order of multipath diversity allowed and thus on the performance of the SSMA system.

To the author's knowledge, no work has been done on the performance of very low rate FEC codes in a non-coherent system operating over indoor multipath radio channels.

3. Simulation Tools Used

To create a model of the proposed system, two software packages were used. The first was SIRCIM (Simulation of Indoor Radio Channel Impulse response Models) and the second was BOSS (Block Oriented Systems Simulator). SIRCIM is an MS-DOS based software package which was used to generate statistically accurate models of different indoor radio channel impulse responses. The data files generated by SIRCIM were then transferred to the UNIX environment where they could be utilized by BOSS. The system simulation was built and executed with BOSS.

Programs in Fortran and C were written to support the BOSS simulation system and for other analyses.

3.1 Simulation of Indoor Radio Channel Impulse response Models (SIRCIM)

The multipath radio channel response profiles used in the simulations were obtained from the SIRCIM software package [6]. This MS-DOS based package generates statistically based impulse responses for different types of indoor wireless channels. Detailed information on SIRCIM can be found in [6,7,29]. A brief overview of the important aspects and features of SIRCIM is given below.

SIRCIM uses statistically based models to produce generic indoor channel impulse responses because neither the number, quality, and placement of potential reflectors or the movement of the transmitter or receiver can be incorporated into a purely deterministic form. The models are generated from 1.35 GHz modulated signals, but the results can be extended for modulated signals up to 4.0 GHz with reasonable accuracy [30]. Real measured channel responses were taken and used to create statistical models which determine the

number, the arrival times, and the amplitudes of distinct multipath components in three different types of indoor environments: open-plan, hard partitioned, and soft partitioned office buildings. The open-plan building is equivalent to a factory or warehouse where there are mostly large open areas with a few large obstacles. The hard partitioned building represents a typical multi-storied office building with many walls made up of concrete or drywall supported by metal studs spaced 16 inches apart. Hallways are 6 to 10 feet wide. The soft partitioned building represents a typical multi-storied office building, but with short (5 feet high) cloth covered dividers used to form cubicles instead of walls. SIRCIM takes the measured channel data and creates multipath channel profiles with the same ensemble and local statistics [7]. A multipath channel profile consists of a set of impulses, where each impulse has a certain power, phase, and relative delay determined by the statistical properties of the channel. The work in this thesis uses the soft partitioned building channel model.

The channel measurements on which the SIRCIM package was based were performed by transmitting a square-pulse, $p_s(t)$, amplitude modulating a 1.35 GHz carrier. It was assumed that the pulse $p_s(t)$ had a short enough duration that there was at most one multipath component and no pulse overlap within a time window equal to this duration. The received power response was thus given by [7]:

$$|h_b(t)|^2 = \Sigma_k(\alpha_k{}^2 p_s{}^2(t-\tau_k)) \qquad (3.1.1)$$

where α_k is the real voltage attenuation factor and τ_k is the delay of the k^{th} path in the channel with respect to the line-of-sight path. This signal was then separated into bins of duration T_{imp} = 7.8125 ns. Within each bin, the α_k values were averaged using a linear interpolation method [7] to obtain A_k. A_k

represents the average path gain at a discrete time T_k, where T_k represents a value equal to kT_{imp}. The resulting impulse power response is [7]:

$$|h_{bd}(t)|^2 = \Sigma_k A^2{}_k \delta(t-T_k) \qquad \delta(t) = \text{impulse function} \qquad (3.1.2)$$

The measurements just described were performed successively at locations separated by 5.5 cm (equal to one quarter of a wavelength of a 1.35 GHz carrier). Thus, there were 19 channel profiles per metre. By doing this, the small scale variation in the fading, the number, and the arrival times of the multipath components could be characterized over a small area. The impulse response profiles at distances between these 5.5 cm spacings could be accurately interpolated by using a cubic spline method [6].

It was found that the time delay spread was almost always less than 500 ns, so this value was set as the maximum duration of each channel profile. Thus, with the channel quantized into 7.8125 ns bins, up to 64 multipath components could be resolvable in each channel profile.

The SIRCIM simulation package also allows for channels with line-of-sight (LOS) or obstructed (OBS) topographies. LOS topographies have a direct path between transmitter and receiver while OBS topographies do not. Different distributions of multipath components arise from the two types of topographies.

Distances between the transmitter and receiver can also be specified in SIRCIM. Obviously, this will affect the received power of the signal. For LOS and OBS topographies in hard partitioned or soft partitioned office environments, T-R separations between 10 m and 25 m are allowed. In open plan building environments, T-R separations between 10 m and 65 m for LOS topographies and between 10 m and 50 m for OBS topographies are allowed. The reduced range for the non-open plan buildings and the OBS topographies is the result of

reduced signal strengths at large T-R separations.

All the aforementioned parameters are factors in determining the channel impulse response. Therefore, a general formula for the power impulse response can be written [7]:

$$|h_{bd}(t, X_l, S_m, D, P_n)|^2 = \sum_k A_k^2(T_k, X_l, S_m, D, P_n)\delta(t - T_k(X_l, S_m, D, P_n)) \qquad (3.1.3)$$

where: t = time
X_l = set of locations within local area of 1m (l=1 to 19)
S_m = LOS or OBS for m = 1 or 2 respectively
D = distance between transmitter and receiver (within ranges stated above)
P_n = the set of measurements obtained for the particular building type (open,hard or soft partitioned; n = 1 to 3 respectively)

The phase of each of the multipath components is computed using geometrical assumptions about the environment. A random phase is given to the initial received component. Then, using the user input parameters of receiver speed and direction of travel, and width of the aisle where the transmitter and receiver are located, SIRCIM calculates the change in distance between the transmitter and the receiver (q' - q) and the instantaneous frequency f_i (accounting for the Doppler shift) for each multipath component. Using these values, the change in phase is calculated using the equation below for each multipath component at each delay T_k and for each small movement of the mobile terminal. Further details can be found in [6].

$$\Delta_{phase} = (q' - q)2\pi f_i / \upsilon \qquad (3.1.4)$$

where υ = signal propagation velocity = 3×10^8 m/s for free space

The complete multipath channel profiles can be written to a file and included in a library of impulse responses for indoor multipath channels.

This thesis uses soft partitioned indoor radio channels with no line-of-sight path between the transmitter and the receiver. For this type of channel model, the following parameters are used [6].

Mean number of multipath components:

$$E[N_p(X,S_2,P_3)] = \text{Uniform } [2,10] \tag{3.1.5}$$

Standard Deviation of $N_p(X,S_2,P_3)$:

$$\sigma_p(S_2,P_3) = 0.4\ E[N_p(X,S_2,P_3)] \tag{3.1.6}$$

Probability of component arrival at a particular delay T_k:

$$\begin{aligned} P_R(T_k,S_2) &= 0.7+T_k/(1000\text{ ns}) && 0\text{ ns} \le T_k < 100\text{ ns} \\ &= 0.8\exp(-(T_k-100\text{ ns})/50\text{ ns}) && 100\text{ ns} \le T_k \le 500\text{ ns} \end{aligned} \tag{3.1.7}$$

Mean large scale path loss exponent:

$$\begin{aligned} n(T_k,S_2) &= 4.8 + T_k/(484\text{ ns}) && T_k < 250\text{ ns} \\ &= 5.3 && 250\text{ ns} \le T_k \le 500\text{ ns} \end{aligned} \tag{3.1.8}$$

Variance of $n(T_k,S_2)$:

$$\sigma_{\text{large-scale}}(T_k,S_2) = 4 + 4\exp(T_k/39\text{ ns})\text{ dB} \tag{3.1.9}$$

Small scale spatial and temporal correlation coefficients = 0.0.

Figure 13 shows an example of a set of channel profiles obtained for a 1 m shift of the mobile terminal.

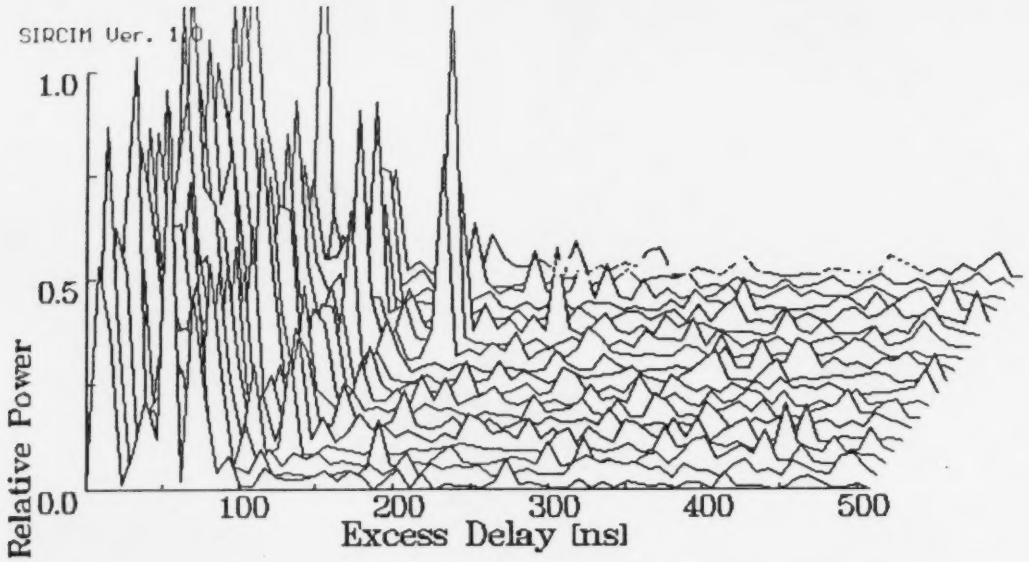


Figure 13. Diagram of a set of soft partitioned office multipath channels.

3.2 Block Oriented Systems Simulator (BOSS)

The simulations were performed using Comdisco's Block Oriented Systems Simulator (BOSS) [5] on SUN SPARC 1+ and SPARC 2 workstations. BOSS performs time domain simulations of systems designed by the user. These systems are constructed by linking components from a library of pre-programmed *blocks* which perform specific functions such as impulse generation, random data generation, digital modulation and many more. The blocks are functional representations of Fortran programs whose inputs and outputs can be interlinked.

BOSS also allows the user to form his own custom function modules. These can be created by linking other modules together in a hierarchical structure to perform the desired function or by writing a Fortran subroutine which performs the desired function and linking it to an empty block which has only inputs and outputs. For the latter, the user must define the inputs and outputs of the custom module and then link them to the inputs and outputs of the Fortran subroutine which performs the desired function. This type of custom module is

called a BOSS primitive.

Complicated initial parameters can be calculated before the simulation starts by using initialization code. The initialization code is a Fortran subroutine which has its inputs and outputs linked with parameters from the system or module used in the simulation. Thus, constants such as filter coefficients can be calculated prior to the simulation.

An example of a simple BPSK system implemented on BOSS is shown below.

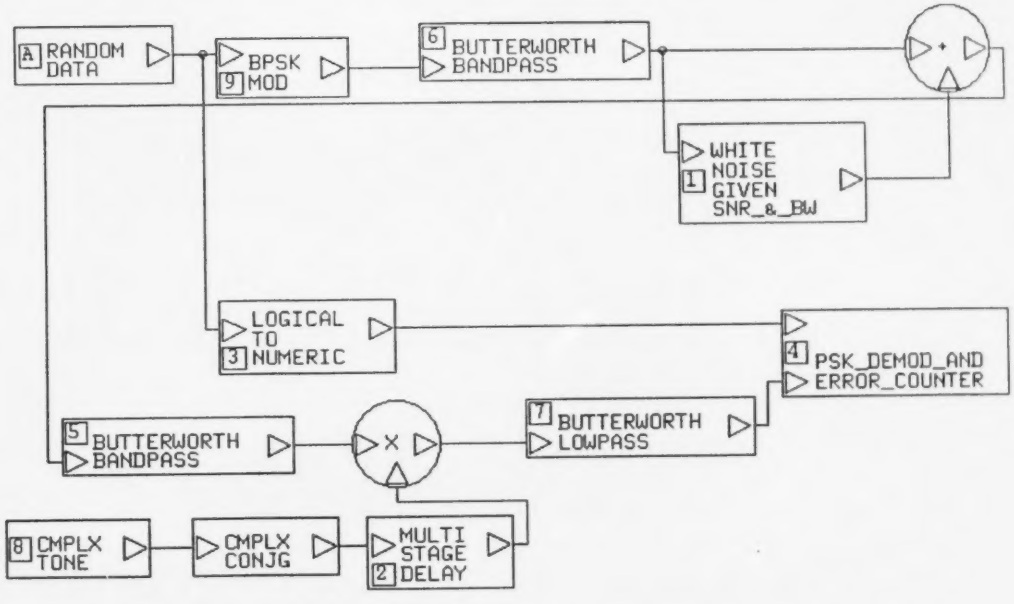


Figure 14. Diagram of a BOSS system.

When running a simulation, BOSS executes each block once every sampling time. This sampling time, which is specified by the user, represents the maximum time resolution of the simulation. The sampling time must be chosen very carefully in BOSS simulations: every periodic waveform must be of

a frequency which divides into the sampling frequency evenly. If it does not, BOSS will change this frequency to the closest value which does.

Different blocks can have different initialization parameters. Changing the values of these parameters will change the constraints or operating conditions of the system being simulated. In our simulation structure, custom BOSS modules were built for encoding, modulation, indoor multipath channel convolution, demodulation, decoding, and performance analysis. Initialization code was written for the encoder and for the complete SSMA system.

Custom BOSS primitives and modules along with brief descriptions and any supporting code are found in Appendix 1.

3.3 Fortran and C Support Programs

Besides the initialization and primitive module codes for BOSS which were written as Fortran subroutines, a few support programs were required. These included programs which generated files which contained the valid orthogonal convolutional codewords and the Gold sequences, a program used to find the normalized mean magnitude of the cross-correlation between partial and complete Gold sequences, and one for computing the error bounds for orthogonal convolutional codes. Other programs were written to find the variance and mean of the data collected for interference analysis, to perform the Kolmogorov-Smirnov goodness-of-fit test of interference data to a Gaussian distribution, and to alter the SIRCIM impulse response data into a more usable format.

Estimates of bit error rates made from analyses of channel profiles were also performed using a Fortran program. This estimation method is discussed in Subsection 4.1.3.

All these programs were written in Fortran except for the one which

generated valid orthogonal convolutional codes which was written in C. These programs can be found in Appendix 2.

4. Overview of the SSMA System and Its Simulation Strategy

4.1 SSMA System

The performance of a direct sequence spread spectrum multiple access system utilizing low rate orthogonal convolutional codes is to be evaluated. The bit error rate and the system capacity are used as performance parameters to determine the effect of different code rates.

4.1.1 System Description

A block diagram of the investigated system is shown in Figure 15.

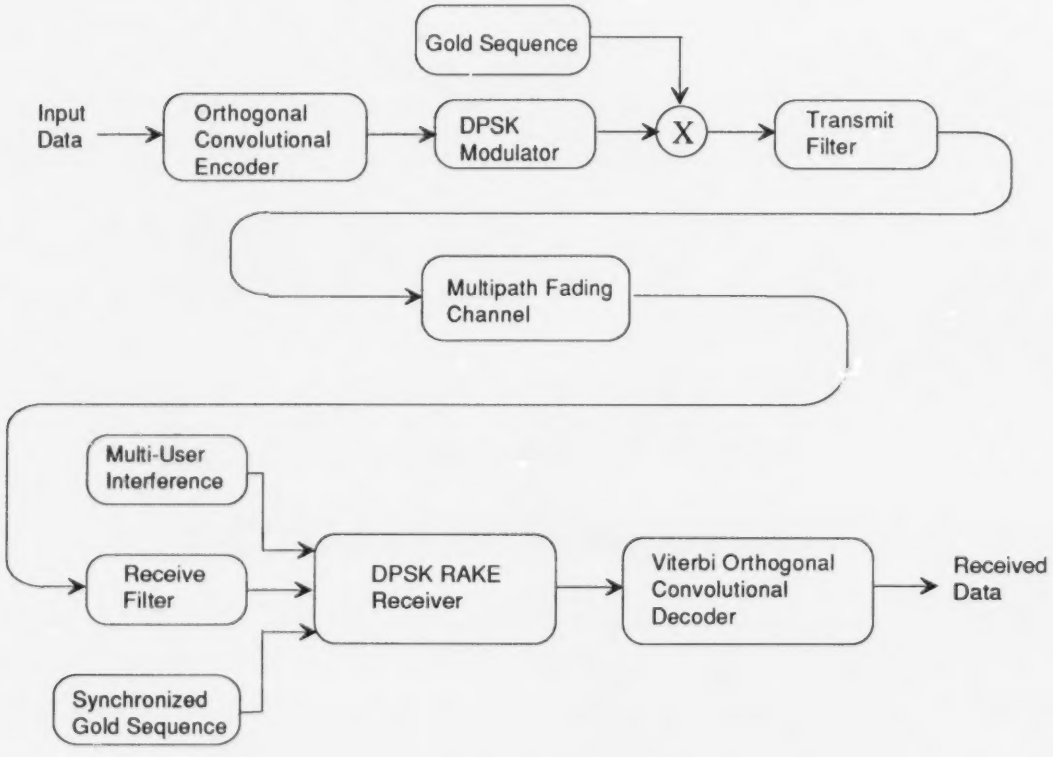


Figure 15. Block diagram of the analyzed SSMA system.

The data rate used in this system is 31.25 kbits/s. This value is used because it is close to the standard 32 kbits/s and its period is an integer multiple of the simulation sampling period of 7.8125 ns. The final spread spectrum chip rate of the system remains fixed at 32 Mchips/s which is 1024 times the data rate. Thus, there are 4 samples per chip time. The FEC code rate is varied between 1, 1/8, 1/32, 1/128 and 1/1024 for different simulation runs, and thus the subsequent spreading of the encoded data varies between 1024, 128, 32, 8 and 1 times respectively.

Orthogonal convolutional codes are used for FEC coding because they have excellent error correcting capabilities and because very low rate codes can be generated very simply by the encoder shown in Figure 6. Also, a fairly simple decoding mechanism known as the Green Machine can obtain the branch metrics serially in the decoding process of these codes (see Subsection 2.2.2 or [19]).

Gold sequences are used to achieve multiple access. The length of these sequences does not change when the code rate changes; it remains a sequence of length 1024. Note though that the length of true Gold sequences is only 1023 chips while 1024 chips are required per data symbol period. This problem is solved by adding one extra chip to the end of all Gold sequences used in the system (see Subsection 2.1.6). It is expected that the mean normalized cross-correlation between spreading sequences would degrade only slightly as a result of this extension. From this point on, whenever Gold sequences are referred to, it will be to these extended sequences rather than to the true ones.

When code rates lower than 1 are used, each encoded symbol is spread by only a fraction of the Gold sequence. For example, if the code rate is 1/32, giving an encoded symbol rate of 1 Mbits/s, the Gold sequence would be required to increase this symbol rate 32 times. The first 32 chips of the Gold

sequence would spread the first encoded symbol, the second 32 chips of the Gold sequence would spread the second encoded symbol, and so on until the thirty-second encoded symbol is spread by the last 32 chips of the Gold sequence, after which this procedure is repeated (see Subsection 2.6.1). Worst-case and average cross-correlations between partial Gold sequences are expected to be worse than those obtained with complete Gold sequences of similar length. However, the number of different Gold sequences is greater for longer sequences, and since the maximum number of multiple access users allowed cannot exceed the number of sequences available, the longer Gold sequences will not impose a hard-limit on the number of users in a realistic system as a result of their length. For the 1024 chip long modified Gold sequences used, there are 1025 distinct sequences. In order to achieve a uniform approach to the simulations while the ultimate capacity of the system was still unknown, Gold sequences of length 1024 were used for all code rates investigated.

The data to be transmitted is first encoded with the appropriate orthogonal convolutional code. It is then DPSK modulated before being spread by a Gold sequence. The spreading is done by multiplication of the DPSK modulated signal and the Gold sequence. Then, the baseband result is passed through a transmit filter. The filter to be used is a third-order Butterworth band-pass filter with a 3 dB bandwidth equal to the chip rate.

The simulation, however, is done at baseband. Thus, a baseband representation of DPSK modulation is implemented and the band-pass filter is represented with a low-pass third-order Butterworth filter with a 3 dB frequency equal to half the chip rate. The signal prior to this low-pass filter consists of square pulses. These square pulses are represented as complex numbers with a magnitude of one. This complex representation allows a phase to be

introduced to the DPSK encoded signal. The phase of these pulses is used to represent a phase offset between the transmitted signal carrier and the demodulating waveform of the receiver.

The transmitted signal propagates through a multipath channel before reaching the receiver. The channels are derived from the SIRCIM software package as described in Section 3.1. Channel models with statistics corresponding to soft partitioned offices with no line-of-sight paths between transmitter and receiver are used. The transmitter and receiver are assumed to be 10 m apart and moving at a velocity of 1 m/s in random directions with respect to one another. The soft partitioned offices have hallways which are 3 m wide. The maximum time delay spread of the channels is 500 ns. To simulate perfect power control, the channel profiles are normalized to contain unit energy when convolved with a square pulse of duration equal to one chip time.

Multi-user interference is added to the signal at this point. This interference is assumed to be Gaussian in nature. This assumption is justified later in this thesis. The variance of the interference is proportional to the number of interferers in the system and the normalized mean magnitude of the cross-correlation between partial Gold sequences. The modelling of the interference is discussed in Section 4.2. Since perfect power control is assumed in this system, the average power levels of the signals from each interferer are the same.

The thermal noise level is normally negligible in comparison to the interference levels so it is ignored in this study. With perfect power control, the signal levels can be kept at a level high enough to make thermal noise effects negligible at the bit error rates of interest.

At the receiver, in order to eliminate out-of-band noise, the composite signal is passed through a Butterworth filter which is identical to the transmit filter. It is then despread with a Gold sequence identical to the one at the

transmitter. This sequence is synchronized to the one in the desired received signal. After despreading, the signal passes through an integrate and dump filter which suppresses the multi-user interference. The result is then DPSK demodulated.

A RAKE receiver [13,31] is used to take advantage of the multipath diversity which results from multipath propagation. The structure of the RAKE receiver for DPSK signals is shown in Figure 16 below.

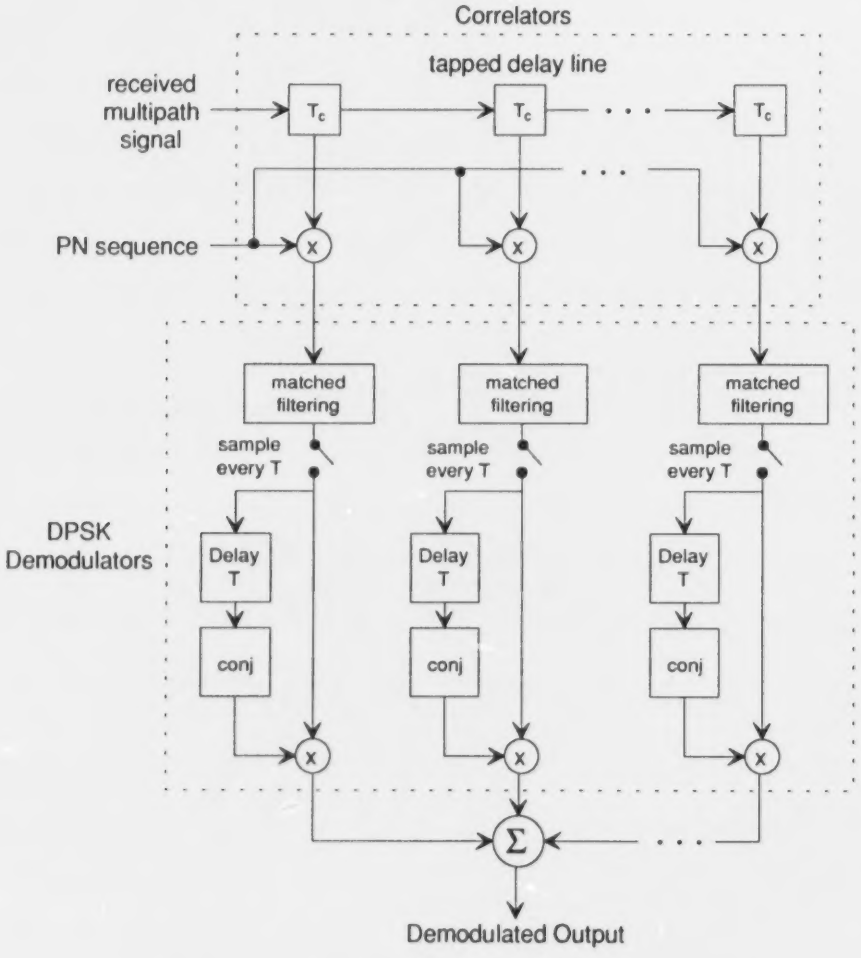


Figure 16. A RAKE demodulator for DPSK signals.

The RAKE receiver consists of two main parts: the correlator and the demodulator. The correlator is made up of a tapped delay line which the multipath signal propagates through and whose outputs are correlated with a despreading PN sequence. The premise of this operation is that if the normalized cross-correlation between the desired signal and the interference (which includes self interference resulting from the multipath propagation) is very low, then the output from each correlator stage will come mainly from only one path of the desired multipath signal. The tap delays are equal to T_c, which is the chip duration. This allows the RAKE receiver to distinguish multipath signals which are separated by a time of T_c or greater. Paths which are closer together are treated as a single path.

The demodulator section consists of standard DPSK demodulation components. Generally, integrate and dump filters are used as the matched filters since square pulses or band-limited square pulses are usually transmitted. The conjugation modules are used to as part of the demodulation process to transform the complex integrate and dump samples into real value outputs.

There are an equal number of demodulator and correlation stages. The outputs of the demodulators are combined with equal weighting given to each stage. Of course, stages operating on lower power signal components will produce smaller outputs.

The diversity order of the receiver is equal to the number of stages in the tapped delay line of the correlator. The maximum diversity order that can be used by a RAKE receiver is given by the equation:

$$L_{max} = \text{floor}[\Delta_{ds}/T_c]+1 \qquad (4.1.1.1)$$

where Δ_{ds} is the channel delay spread, T_c is the duration of a chip, and the floor[x] function produces the largest integer less than or equal to x. In the

system being studied, the maximum diversity order would be floor[500×10^{-9} / ($1/(32\times10^{-6})$)] + 1 = 17. However, only third order diversity is actually used because it was determined that the channel models generally have most of their energy concentrated within a 90 ns delay region. Trying to gain diversity by using multipath components with very low power would not improve performance and could actually be counter productive [13].

In this system, code synchronization, Gold sequence synchronization, and optimal signal sampling are assumed. Code synchronization means that the beginning of each codeword of the received signal has been found and locked onto. Gold sequence synchronization means that the Gold sequence which spreads the data in the received signal and the Gold sequence which performs the despreading at the correlator of the RAKE receiver are matched so that the received signal will be properly despread. The despread signal is sampled at its optimum point. This optimal point is the point where the largest average eye opening is detected at the output of the integrate and dump filters of the RAKE receiver. This models the behavior of a receiver which actively adapts to changes in the signal it is trying to acquire. The output of the RAKE receiver is subjected to hard-decision binary detection and fed to the FEC decoder.

The FEC decoder performs hard-decision Viterbi decoding on this binary sequence. The decoding path length is truncated to 5 times the code's constraint length (see Subsection 2.2.2 for details). The decoded output is the decoder's best estimate of the transmitted sequence from its received sequence. This output is compared with the actual transmitted data so that a bit error rate value can be obtained.

The BOSS block diagram of the SSMA system is shown in Figure 17.

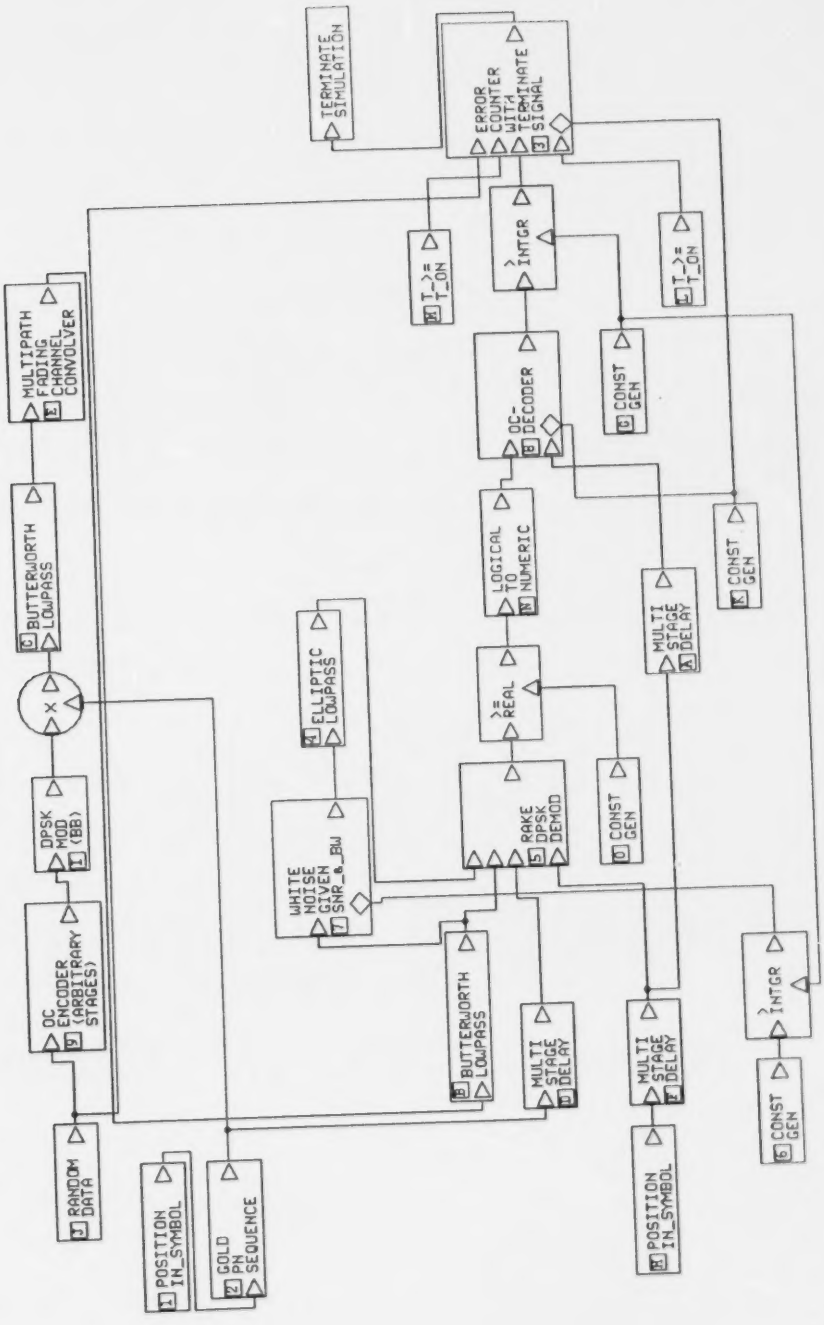


Figure 17. BOSS implementation of the SSMA system.

4.1.2 Full Simulation Strategy

Bit error rates are collected by simulating the system with different channel profiles, different code rates, and different interference levels. Ten different channel profiles are used for the systems using code rates of 1/1024, 1/128, and 1/32. Each simulation is run until 3 errors are detected. This gives a 90% confidence level that the BER found in the simulation is within a factor of approximately 2.5 of the actual BER [32]. Sixteen different channel profiles are used for the systems using code rates of 1/8 and 1. Each of these simulations are run until 4 errors are detected. This gives a 90% confidence level that the BER found is within a factor of approximately 2 of the actual BER. The extra channel profiles used and the extra bit error detected in the higher code rate simulations are possible because the systems are simpler and require less computation time.

4.1.3 BER Estimates Using Error Bounds and Channel Analysis

Because of the very long simulations required using the full simulation technique described in Subsection 4.1.2, two other methods are used to estimate the system BER.

The first method finds the encoded symbol error rate p at the output of the RAKE DPSK receiver and substitutes the value into equation (2.2.3.11) to bound the decoded BER. We will call it the error bounding method. Since the symbol rate is much higher than the data bit rate (except for the uncoded system) and the symbol error rate is expected to be quite high, this estimate can be obtained much faster than with the full simulation method. Obviously, this method is useful only for studying SSMA systems which utilize coding. This method is particularly useful for analyzing systems which use very low rate codes and which have very low BERs.

The simulation model is adjusted by removing the OC-DECODER and its supporting modules. The ERROR COUNTER WITH TERMINATE SIGNAL takes the output of the RAKE DPSK DEMOD module (after it has been converted to logical form) as its test signal and the output of the OC ENCODER (ARBITRARY STAGES) as its reference signal. This model is shown in Figure A15 in Appendix 1.

The second method for approximating the BER is used to estimate the bit error rate of the uncoded SSMA system. It involves analysis of the channel impulse response profile. We will call it the channel analysis method. The program which performs this analysis is called PROG_CHANNEL_ANAL and can be found in A2.vii.

First, a representation of the signal seen by the receiver must be made. To do this, the result of a single square pulse $p_{chip}(t)$, of duration equal to the chip time, convolved in time with the channel impulse response profile must be found. The actual transmitted pulse in the simulated SSMA system is not perfectly square. It is in fact a square pulse that has been passed through a low-pass filter. However, for simplicity, the ideal square pulse is used in this analysis. The received signal can be expressed as:

$$r(t) = \sum_{k=0}^{64} A_k p_{chip}(t - T_k) \qquad (4.1.3.1)$$

where A_k is the gain of the k^{th} path in the channel profile
T_k represents a value of k(7.8125) ns
$p_{chip}(t)$ is a square pulse of duration equal to a chip time

Because the chip consists of four samples and the channel model consists of 65 samples, when the two are convolved, the result is a signal, r(t), of 68 samples.

Since the resolution of the RAKE receiver is equal to one chip time, the energy from the received signal which can be used by the j^{th} stage of the receiver can be expressed in terms of the energy of the chip which is synchronized to the j^{th} stage:

$$E_{chip,j} = \frac{\alpha^2 E^2}{E} = \frac{\left| \int_{\tau_j}^{\tau_j + T_c} r(t)u^*(t)dt \right|^2}{\left| \int_0^{T_c} u(t)u^*(t)dt \right|} \approx \frac{\left(\frac{T_{imp} \sum_{k=n_j}^{n_j+3} r(T_k)}{4T_{imp}} \right)^2 (4T_{imp})^2}{4T_{imp}} \qquad (4.1.3.2)$$

where E is the unattenuated chip energy = $T_c = 4T_{imp}$

α is the attenuation factor

τ_j is a time offset for receiving the signal for the j^{th} stage

T_c is the chip time

$u(t) = u^*(t) = 1$ is the ideal square pulse matched filter waveform

T_{imp} is the time between impulses in the channel profile

$T_{imp} = 7.8125$ ns

n_j is the point where the j^{th} stage of the receiver begins its integrate and dump function

Remembering that $T_c = 4T_{imp}$, $n_3 = n_2 + 4 = n_1 + 8$, thus explaining the limits on the summation. The value of n_1 is found so that the sum of $E_{chip,1}$, $E_{chip,2}$ and $E_{chip,3}$ is maximized for the third order RAKE receiver used in this study. In other words, the signal is to be sampled where the largest average eye openings of the three integrate and dump filters of the RAKE receiver are found.

An estimate of the total energy of the received signal E_{sig} can be made by summing the power of r(t) and multiplying it by T_{imp}. This estimate makes the assumption that the power level between two discrete signal points is equal to that at the first of the two signal points. If perfect power control is present, E_{sig} is also equivalent to the energy of each interfering signal seen by the receiver over one chip time. This is because the sum of each group of impulse powers over

one chip time represents the energy from an interfering signal which has been suitably delayed. This energy is given as:

$$E_{sig} = T_{imp} \sum_{k=0}^{67} |r(T_k)|^2 \qquad (4.1.3.3)$$

The average effective multi-user interference energy can thus be determined as:

$$E_{int} = 0.5n(xcorr)(E_{sig}) \qquad (4.1.3.4)$$

where n is the number of interfering users
xcorr is the normalized mean magnitude of the cross-correlation between partial or complete Gold sequences
the factor of 0.5 results from the assumed 50% voice activation factor

An explanation on the derivation of xcorr is given in Section 4.2 where the interference power is discussed in detail. The experimentally determined values of the mean magnitudes of the cross-correlations are given in Subsection 5.1.2.

The average intersymbol self-interference for stage j of the receiver can also be determined as:

$$E_{self-int,j} = xcorr(E_{sig} - \sum_{k=n_j}^{n_j+3} |r(T_k)|^2) \qquad (4.1.3.5)$$

The signal to interference ratio on the j^{th} stage can be expressed as:

$$SNR_j = \frac{E_{chip,j}}{E_{int} + E_{self-int,j}} \qquad (4.1.3.6)$$

After finding the signal to interference ratio for all three stages of the third order RAKE receiver, we can calculate the probability of an encoded symbol error at its output. This is done by treating the RAKE receiver as a multi-channel DPSK receiver. In other words, it is a receiver which receives L = 3 DPSK

signals over 3 different channels, each with different signal to interference ratios, and combines them equally. An analysis for this receiver is given in [13]. The bit error rate can be expressed as:

$$P_b = \frac{e^{-\gamma_s}}{2^{2L-1}} \sum_{i=0}^{L-1} c_i \gamma_s^i \qquad (4.1.3.7)$$

where $L = 3$

$$c_i = \frac{1}{i!} \sum_{k=0}^{L-1-i} \binom{2L-1}{k}$$

$$\gamma_s = \sum_{j=1}^{L} SNR_j$$

4.2 Interference Modelling System

A system implemented on BOSS which generates multi-user interference is used to generate data samples which can be analyzed in order to make a statistically accurate baseband model of the interference. The output of this system is the sum of many spread spectrum signals after each has propagated through its own multipath channel. The BOSS block diagrams of this system and its internal modules are in A1.viii.

For each interfering user, a random binary signal at the chip rate of 32 Mchips/s is generated. The random binary signal represents the direct-sequence spread signal. It is reasonable to assume that this random signal can represent a data sequence that is encoded and spread by a Gold sequence since the latter is quite random also for the short time intervals used in the simulations. Each signal is offset by a random phase which is equivalent to the phase of the carrier, and delayed by a random time period of up to 1 chip time. This simulates the asynchronous nature of the interference. To account for the user activation factor, each user's signal is squelched randomly 50% of the time.

The final output signal from each interfering user is convolved with its own

multipath channel profile. Perfect power control is implemented by normalizing the energy contained in each multipath channel profile. The resulting signals from all the interfering users are then summed and passed through a third-order Butterworth low-pass transmit filter to form the composite interference signal that the receiver of interest detects. In a real system, this transmit filter would be before the multipath channel, but since both are linear filters, the sequence of the filters is not important.

The interference signal passes through a receive filter which is identical to the transmit filter and is then multiplied by a Gold despreading sequence. The complex value of the resulting signal is written to a data file which can then be analyzed to determine its frequency spectrum and approximation to a Gaussian distribution. This is used to verify if the interference can be represented as white Gaussian noise. Results of this analysis are given in Subsection 5.1.1.

To estimate the signal-to-noise ratio that results when there are j-1 interfering users, the effect of the cross-correlations between Gold sequences must be taken into account. The aperiodic cross-correlation between two sequences can be defined as [3]:

$$\begin{aligned} C_{k,j}(l) &= \sum_{j=0}^{N_c-1-l} a_j^k a_{j+l}^j && 0 \le l \le N_c - 1 \\ &= \sum_{j=0}^{N_c-1+l} a_{j-l}^k a_j^j && 1 - N_c \le l < 0 \\ &= 0 && |l| \ge N_c \end{aligned} \tag{4.2.1}$$

where a_j^k is the j^{th} chip of the k^{th} user and N_c is the length of the sequence. The value N_c can equal the length of the entire sequence (if complete sequences are used for spreading) or the length of the partial sequence (for partial sequence spreading).

The continuous-time partial cross-correlation functions are defined as [3]:

$$R_{k,i}(\tau_k) = C_{k,i}(l-N_c)T_c + (\tau_k - lT_c)[C_{k,i}(l+1-N_c) - C_{k,i}(l-N_c)] = \int_0^{\tau_k} a^k(t-\tau_k)a^i(t)dt \quad (4.2.2)$$

$$\hat{R}_{k,i}(\tau_k) = C_{k,i}(l)T_c + (\tau_k - lT_c)[C_{k,i}(l+1) - C_{k,i}(l)] = \int_{\tau_k}^{T} a^k(t-\tau_k)a^i(t)dt \quad (4.2.3)$$

where τ_k is the random delay of the k^{th} user's signal arriving at the receiver, T_c is the duration of one chip, T is the duration of the unspread symbol, and $a^k(t)$ is the continuous version of the k^{th} user's PN sequence. The division of the partial cross-correlation function into two halves is necessary in order to account for unsynchronized nature of transmissions in the system being studied. Because user transmissions are not synchronized, the interfering signal's data symbols may not be aligned with that at the receiver. Therefore, a change in state of the data symbol would change the polarity of part of the interferer's spreading sequence. This requires that the partial cross-correlation functions consist of two parts: one to account for the first part of the interference sequence, and another to account for any change in polarity of the interference sequence resulting from a change in state of its data symbols.

Assuming perfect power control, we can now write the average interference power, normalized to the i^{th} user's received signal power, resulting from the k^{th} user's signal on the i^{th} user as:

$$P_k = \frac{1}{T} E[|x^k_{m-1}R_{k,i}(\tau_k) + x^k_m \hat{R}_{k,i}(\tau_k)|] \quad (4.2.4)$$

where $x^k_m \in \{-1,+1\}$ and represents the m^{th} symbol from the k^{th} user. The mean is taken over all m and τ_k uniformly distributed between 0 and T.

The above equations account for misalignments between the starting points of different PN sequences, but they assume alignment between chip

edges. A more accurate estimate of cross-correlations between unsynchronized SSMA users should take into account unaligned chip edges. To do this, the following modifications are performed.

$$
\begin{aligned}
C_{k,i}(l,\Delta) &= \sum_{j=0}^{N_c-1-l} a_j^k[(1-\Delta)a_{j+l}^i + \Delta a_{j+l+1}^i] && 0 \le l \le N_c - 1 \\
&= \sum_{j=0}^{N_c-1+l} a_{j-l}^k[(1-\Delta)a_j^i + \Delta a_{j+1}^i] && 1-N_c \le l < 0 \\
&= 0 && |l| \ge N_c
\end{aligned}
\tag{4.2.5}
$$

$$
\begin{aligned}
R_{k,i}(\tau_k,\Delta) &= C_{k,i}(l-N_c,\Delta)T_c + (\tau_k - lT_c)[C_{k,i}(l+1-N_c,\Delta) - C_{k,i}(l-N_c,\Delta)] \\
&= \int_0^{\tau_k} a^k(t-\tau_k)a^i(t)dt
\end{aligned}
\tag{4.2.6}
$$

$$
\begin{aligned}
\hat{R}_{k,i}(\tau_k,\Delta) &= C_{k,i}(l,\Delta)T_c + (\tau_k - lT_c)[C_{k,i}(i+1,\Delta) - C_{k,i}(l,\Delta)] \\
&= \int_{\tau_k}^{T} a^k(t-\tau_k)a^i(t)dt
\end{aligned}
\tag{4.2.7}
$$

$$
P_k = \frac{1}{T}E[|x_{m-1}^k R_{k,i}(\tau_k,\Delta) + x_m^k \hat{R}_{k,i}(\tau_k,\Delta)|] \tag{4.2.8}
$$

where Δ is a value between 0 and 1 which is a measure of the misalignment of the chips.

Because average power control is implemented, each interferer's average power will be identical. However, the exact level of interference that it contributes depends upon the cross-correlation properties of its specific spreading sequences. Therefore, an average interference power P contributed by each user can be found by taking the mean of P_k over all τ_k, Δ, m, k, and i where $k \neq i$, thus averaging over many different sequences. Since P_k is already normalized to the power of the desired received signal, so is the value of P. Thus, it is equivalent to the factor xcorr of equations (4.1.3.4) and (4.1.3.5).

$$P = xcorr = E[P_k] = \frac{1}{T} E[|x^k_{m-1} R_{k,i}(\tau_k, \Delta) + x^k_m \hat{R}_{k,i}(\tau_k, \Delta)|] \qquad (4.2.9)$$

The source code PROGRAM_CROSS_CORRELATION_FINDER listed in A2.ii was written to estimate the value of P. Δ values of 0, 0.2, 0.4, 0.6, and 0.8 and τ_k values equal to all multiples of T_c from $-1023T_c$ to $1023T_c$ were used in the estimate. Numerous values of k and i were tested. The values of x^k_m and x^k_{m-1} had equal probability of being +1 or -1. The operation of the program is described below and the results are given in Subsection 5.1.2.

The program finds the mean partial or complete cross-correlations between Gold sequences of length 1024 as follows:

1. All the Gold sequences to be used in the analysis are read into memory.
2. One sequence is taken as the reference sequence and the rest are labelled as test sequences.
3. The reference sequence is divided into $1024/N_c$ partial sequences.
4. The reference sequence is aligned with one test sequence.
5. The magnitudes of the partial cross-correlations between the partial sequences are found and stored.
6. Step 5 is repeated, but with the test sequence shifted by intervals of 0.2 chip times until the test sequence is again aligned with the reference sequence.
7. Steps 2 through 6 are repeated until all the Gold sequences have been paired together once.
8. Steps 2 through 7 are repeated, but this time, adjacent symbols which are spread by the test sequence are assumed to have opposite states. That is, x^k_m and x^k_{m-1} from (4.2.9) equal +1 and -1 respectively.

9. The mean of the partial cross-correlation magnitudes is found.
10. The mean is divided by the partial sequence length in order to get the normalized mean partial cross-correlation magnitude xcorr.

The received signal to multiple access interference ratio of a system with j-1 interfering users can be given as:

$$SIR = \frac{1}{0.5\sum_{k=1}^{j-1} P_k} = \frac{2}{(j-1)P} \qquad (4.2.10)$$

Note that this ratio is not equivalent to E_b/N_o because not all of the desired signal's received power can be used in detecting the signal because of the multipath propagation and delay spread. In fact, many of the delayed paths act as additional interference. The factor of 0.5 in (4.2.10) results from the assumed 50% voice activation factor.

5. Experiments and Results

5.1 Interference Modelling

Following the procedures described in Section 4.2, an analysis of the behavior of power-controlled multi-user interference was performed. The cumulative distribution function and the frequency spectrum of this interference were observed to characterize its behavior. The actual power of this interference which affects detection of the desired SSMA signal was determined from the number of interferers present and the mean normalized cross-correlation of the partial or complete Gold sequences. The results of experiments which characterize the interference and the mean normalized cross-correlations are given in the following sections. These results were used to determine the interference levels in the simulations of the SSMA system.

5.1.1 Statistical Characterization of Interference

Samples of multi-user interference were accumulated for several different numbers of interferers. Approximately 25000 sample points were taken from each set of samples and used in the Kolmogorov-Smirnov test to establish its relationship to the Gaussian distribution. The means and variances of these samples were found using the code PROGRAM_STAT_ANALYSIS in the Appendix A2.iv. Then, using the code PROGRAM_KS_TEST (A2.v), the Kolmogorov-Smirnov one-sided goodness-of-fit test [33,34] was performed to compare the sample data with a Gaussian distribution with the same mean and variance.

The one-sided Kolmogorov-Smirnov test (KS test) is a nonparametric test for differences between two continuous cumulative distribution functions (cdf): one from an observed data set and the other from a hypothesized distribution.

The absolute value of the maximum difference between the two cdf's is called the KS statistic. It can be calculated as:

$$D_n = \max |F(x) - S_n(x)| \qquad (5.1.1.1)$$

where $S_n(x)$ is the empirical cdf of a sample of size n of the random variable x and F(x) is the cdf of the hypothetical distribution.

The distribution of the KS statistic D_n is used to establish the significance level of the test. If the null hypothesis states that the two distributions are of the same type, then the significance level gives the probability that an error is made if the null hypothesis is rejected (a type 1 error, see [33]). The significance level for the KS test is calculated as [35]:

$$\text{Probability}(D_n > D_{n,\text{observed}}) = Q_{KS}(\sqrt{n} D_{n,\text{observed}}) \qquad (5.1.1.2)$$

where

$$Q_{KS}(\chi) = 2\sum_{j=1}^{\infty} (-1)^{j-1} e^{-2j^2\chi^2}$$

Intuitively, due to the nature of the physical mechanism through which interference is generated, the interference samples are expected to be distributed in a Gaussian fashion. Thus, a fairly low value of the level of significance obtained from (5.1.1.2) (above a threshold value of say 0.05) would be considered reasonable to not reject the null hypothesis [36]. The results of the test are given in Table 3 and show that the interference samples cannot be rejected as being Gaussian distributed because the values of the level of significance obtained from (5.1.1.2) considerably exceed the 0.05 threshold.

The frequency spectrum of the interference was also examined to determine if the interference could be modelled as white noise. Figure 18 shows the frequency spectrum of the interference seen at the receiver after filtering but before despreading. The interference is from the transmissions of 9 interfering

# Interferers	Mean of Interference Samples	Variance of Interference Samples	Level of Significance of KS Test
9	-4.63×10^{-3}	2.96	0.89
19	-1.01×10^{-2}	7.12	0.91
29	2.79×10^{-2}	11.66	0.91
39	-1.77×10^{-2}	14.85	0.95
49	1.54×10^{-3}	19.20	0.93

Table 3. Results of Kolmogorov-Smirnov goodness-of-fit test for multiple access interference and Gaussian distribution.

users. This spectrum looks very much like that resulting from a 32 Mchips/s pulse although it is flatter at lower frequencies. This is expected because the effect of the different multipath channels, which would cause notches in parts of

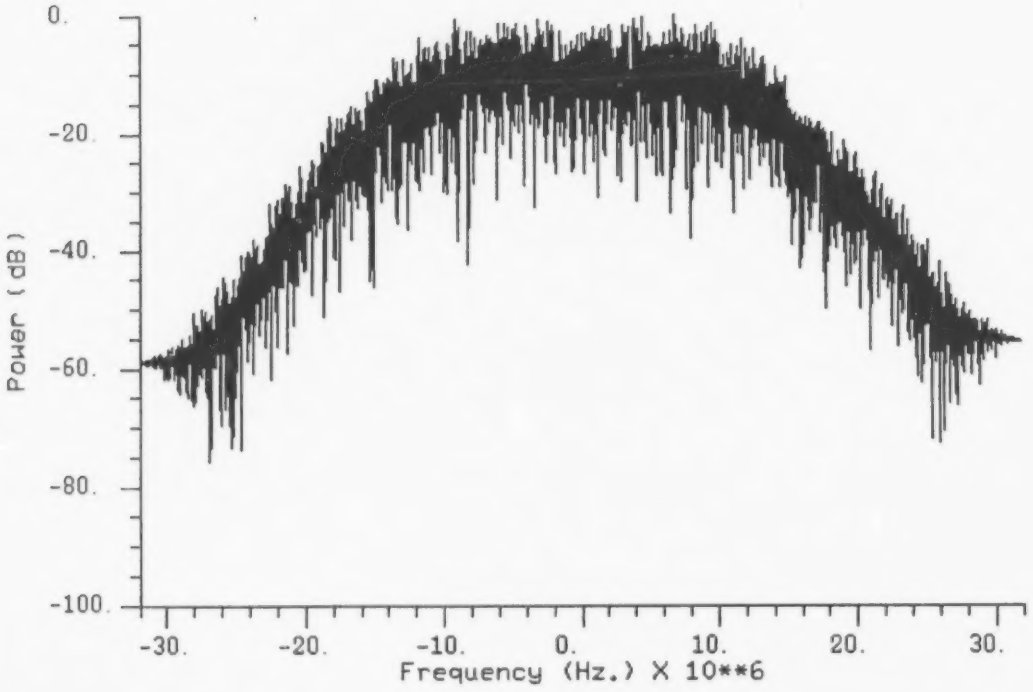


Figure 18. Frequency spectrum of interference before despreading.

the spectrum, has been averaged out. Figure 19 shows the frequency spectrum of the same signal after it is multiplied by the despreading sequence. The spectrum is now almost white. That is, the power is distributed quite evenly throughout the whole 64 MHz spectrum. Some slight power attenuation is evident at the higher frequencies.

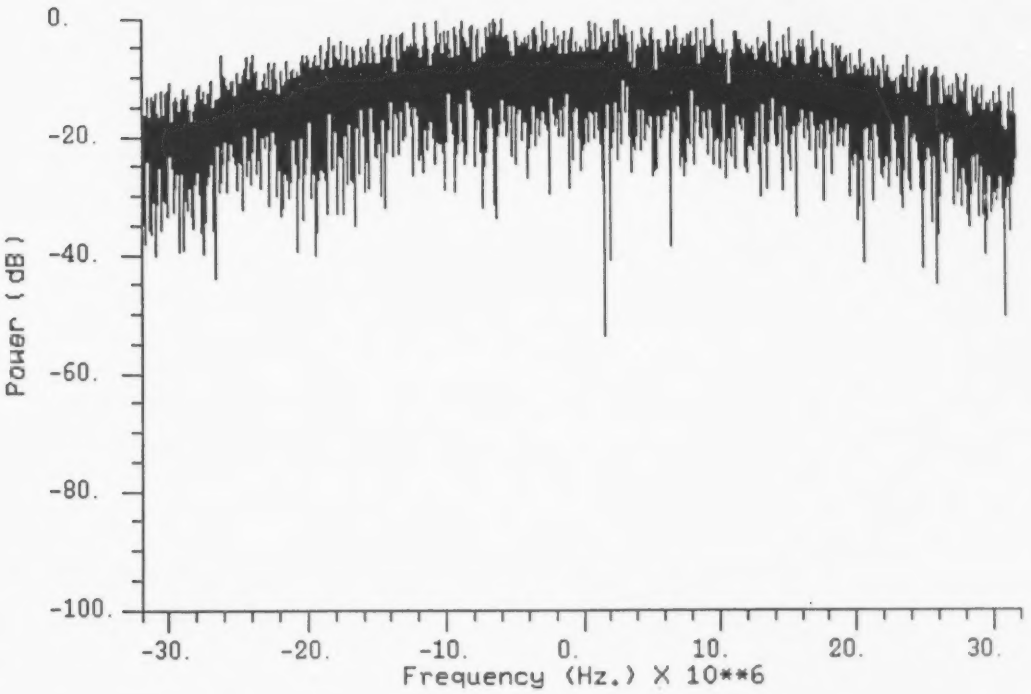


Figure 19. Frequency spectrum of interference after despreading.

This shows that it is reasonable to model the despread interference as white Gaussian noise. Such an assumption simplifies the simulation of the system immensely.

5.1.2 Cross-Correlation of Partial and Complete Gold Sequences

The total power of the interference that influences the detection capability of the desired signal depends not only on the number of interferers, but also the length and cross-correlation of the SSMA spreading sequences. The higher the normalized cross-correlation, the more interference will result. The analysis of the signal to multiple access interference ratio is given in Section 4.2.

The number of sequences of length 1024 (which each contain $1024/N_c$ partial sequences) required in the analysis before convergence to a mean partial sequence cross-correlation magnitude was achieved proved to be quite small. Preliminary runs showed that experiments using only 3 sequences already showed convergence. The results shown here for partial sequence lengths of N_c = 1024, 512, 256, 128, 64, and 32 used only 10 sequences in the analysis. This means that 10, 20, 40, 80, 160, and 320 partial sequences, respectively, were used in finding the mean. For the partial sequence lengths of N_c = 16, 8, 4, and 2, only 3 sequences were used in determining the mean magnitude of the cross-correlation. This means that 192, 384, 768, and 1536 partial sequences, respectively, were used in finding the mean. The mean partial cross-correlation magnitudes for sequence lengths of 1 can be calculated analytically to be 0.75, which can be used to verify that the program works correctly.

The means of the partial cross-correlation magnitudes are given in numerical form in Table 4. The results, normalized to the partial sequence length, are given in graphical form in Figure 20. The normalized values can be interpreted as the percentage of the desired signal's total energy that each interfering user contributes at the receiver. These values are thus equivalent to the value xcorr in equation (4.1.3.4) and (4.2.3.5) or the value of P in (4.2.9).

Partial Sequence Length	Mean Partial Cross-Correlation Magnitude	Maximum Observed Partial Cross-Correlation
1	0.73	1
2	0.86	2
4	1.26	4
8	1.80	8
16	2.57	16
32	3.32	26
64	5.05	40
128	7.36	58
256	10.39	74
512	14.41	106
1024	19.78	142

Table 4. Means and maximums of the cross-correlation magnitudes between partial and complete Gold sequences.

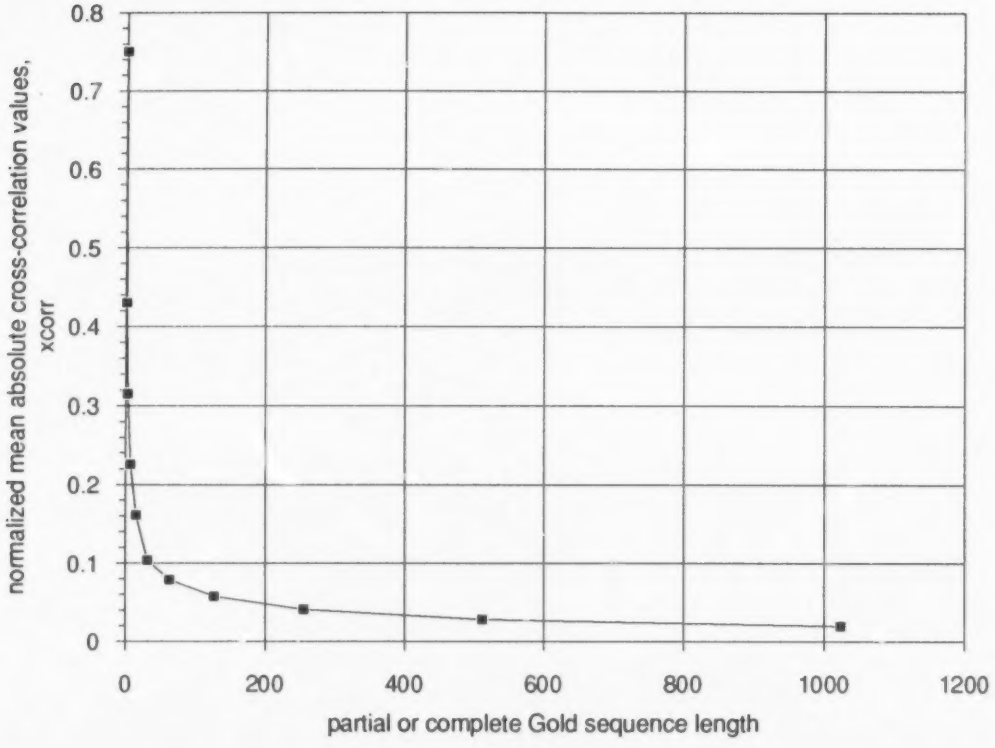


Figure 20. Graph of normalized mean cross-correlation magnitudes between partial and complete Gold sequences.

5.2 System Performance

The results of the SSMA simulations are given in the following sections. Bit error rate results are given for systems with up to 70 users. For voice communications, BERs $\leq 3\times10^{-2}$ are required according to the IS 54-B TDMA digital cellular standard [8]. It is assumed that the 31.25 kbits/s voice code rate used in this system is able to provide similar voice quality at the same BER. Comparisons between the SSMA systems using different code rates are made at BER values of 3×10^{-2}, 1×10^{-3}, and 1×10^{-4}.

Most of the BER values were determined from full simulations of the SSMA systems. However, because the run times of the full simulation method were extremely long when trying to obtain low BER results, some of the values for coded SSMA systems were estimated using the error bounding method after it was verified that it was fairly accurate. The results are presented in the Subsections 5.2.1 and 5.2.2.

The channel analysis method was tried for the uncoded SSMA system. Its accuracy when compared to the full simulation method was found to be good. These results are presented in Subsection 5.2.2.

5.2.1 BER Estimates Using Full Simulation

Figure 21 shows the performance curve for the SSMA system using no FEC coding. This system can be used as a benchmark to compare the other systems. The maximum number of users for BER $\leq 3\times10^{-2}$ was 26. For BER $\leq 1\times10^{-3}$, the maximum number of users was reduced to approximately 14. For BER $\leq 1\times10^{-4}$, the number was only 9.

Figure 22 shows the performance curve for the SSMA system using an orthogonal convolutional code of rate 1/8. The maximum number of users for

BER $\leq 3\times10^{-2}$ was approximately 32. For BER $\leq 1\times10^{-3}$, this number was approximately 18. The system capacity for BER $\leq 1\times10^{-4}$ was not determined using the full simulation method because of time constraints. It was instead analyzed using the error bound method (Subsection 5.2.2). This system offered marginal improvement over the benchmark system with code rate 1.

Figure 23 shows the performance curve of the SSMA system using an orthogonal convolutional code of rate 1/32. Significant improvement over the two previous systems was found. For a BER $\leq 3\times10^{-2}$, the maximum number of users was approximately 48. For BER $\leq 1\times10^{-3}$, the maximum number of users was approximately 30. Again, the system capacity for BER $\leq 1\times10^{-4}$ was not found using the full simulation method, but was analyzed using the error bound method instead.

Figure 24 shows the performance curve of the SSMA system using an orthogonal convolutional code of rate 1/128. For BER $\leq 3\times10^{-2}$, the maximum number of users was approximately 50. For BER $\leq 1\times10^{-3}$, the maximum number of users was approximately 36, and for BER $\leq 1\times10^{-4}$, the maximum number of users was approximately 29. This system performed the best out of all that were simulated.

Figure 25 shows the performance curve of the SSMA system using an orthogonal convolutional code of rate 1/1024. All the spreading in this system was done by the FEC coding so the Gold sequences simply performed a scrambling operation on the encoded symbols. The average performance of this system was extremely poor at the system capacities tested. The range of performance over different channels was also very large. At a system capacity of 30, 60% of the channels simulated provided BERs $\leq 3\times10^{-2}$ among which 66% provided BERs $\leq 1\times10^{-3}$. However, the other channels simulated produced BERs close to 0.5 which resulted in the very high average BER.

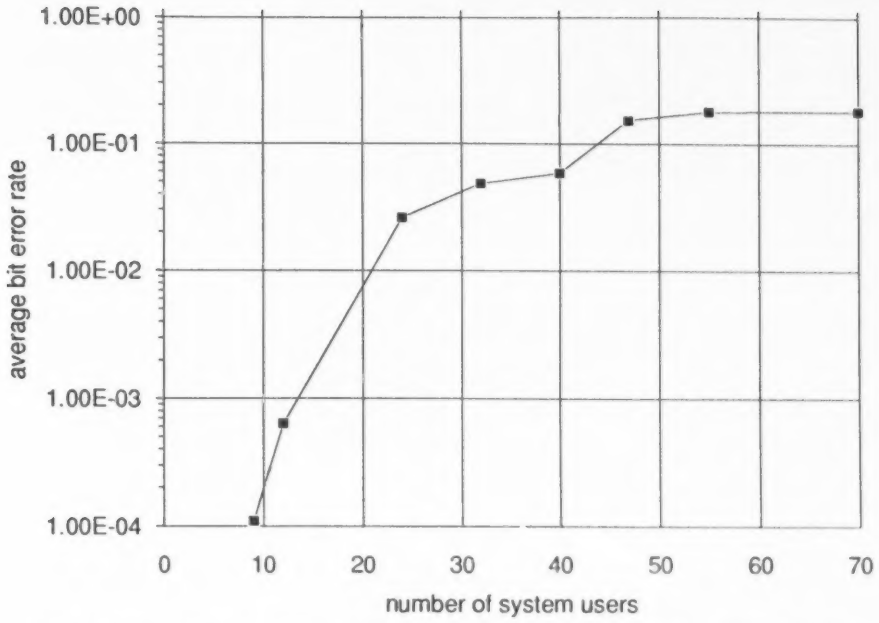


Figure 21. Performance of the SSMA system with no FEC coding.

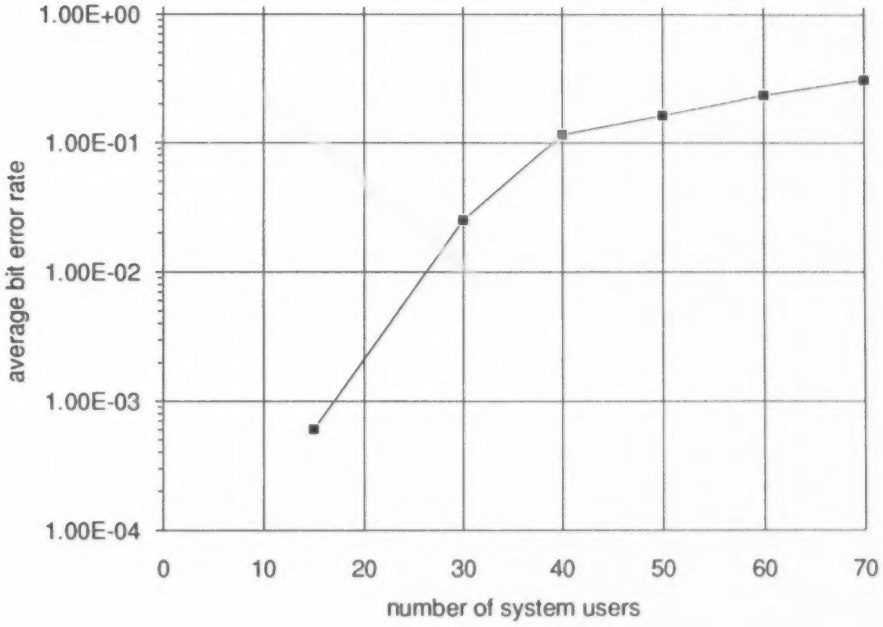


Figure 22. Performance of the SSMA system with code rate 1/8.

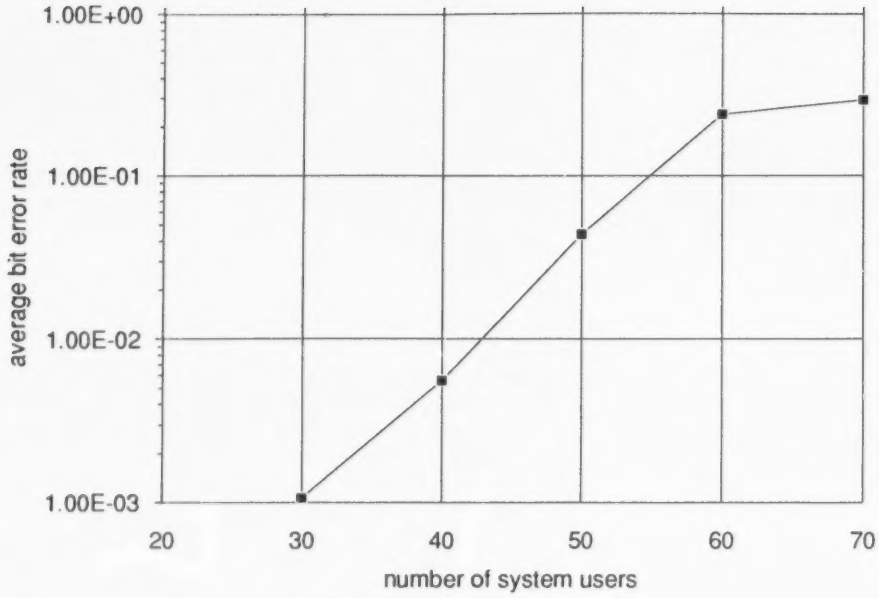


Figure 23. Performance of the SSMA system with code rate 1/32.

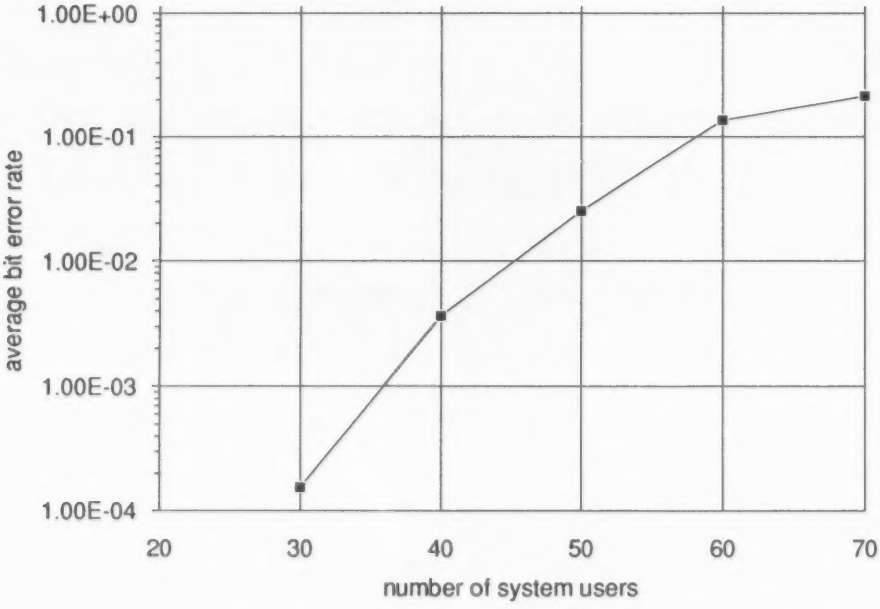


Figure 24. Performance of the SSMA system with code rate 1/128.

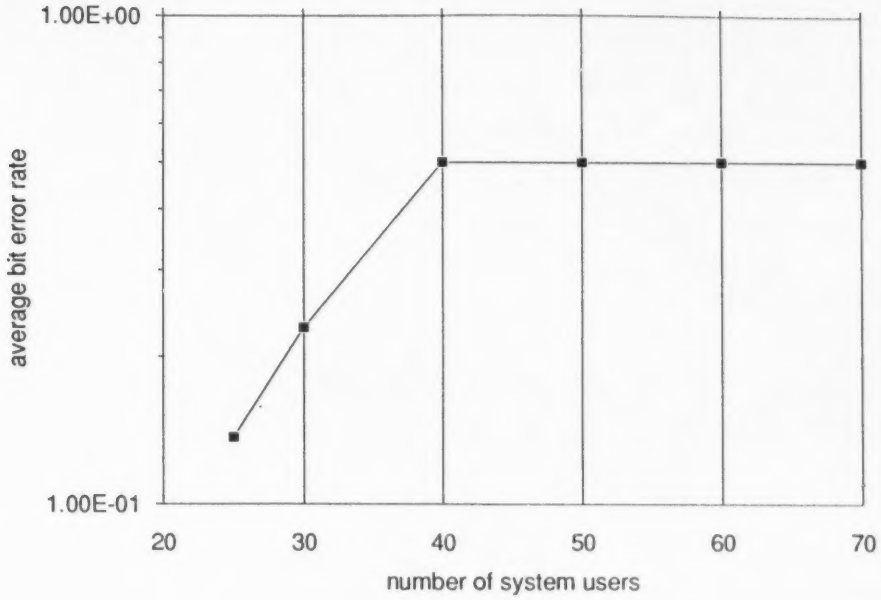


Figure 25. Performance of the SSMA system with code rate of 1/1024.

5.2.2 BER Estimates Using Error Bounds and Channel Analysis

The results of the error bounding method could only be exploited when the encoded symbol error rate was below a certain limiting value which is different for each code rate. Figure 26 shows the graph of the data bit error probability P_b versus the symbol error probability p derived from the bound given by equation (2.2.3.11) for an orthogonal convolutional code of rate 1/128 (constraint length K = 7).

From Figure 26, it is evident that the bound on P_b is extremely loose and therefore useless when symbol error rates approach p = 0.41. Fortunately, this region is of interest only when investigating systems with BER > 5×10^{-3}. Figures 27 through 29 show similar graphs for codes with K = 3, 5, and 10 respectively.

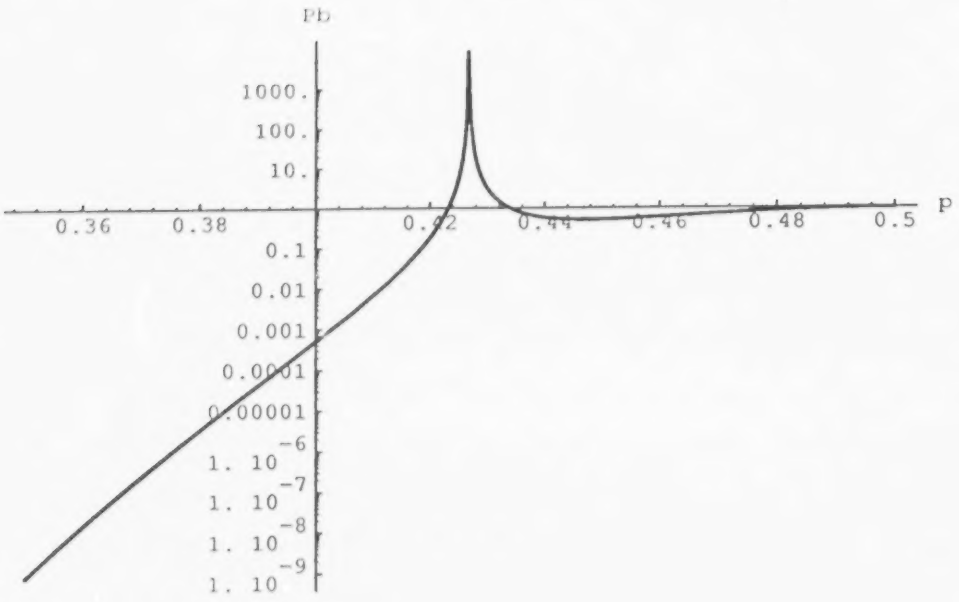


Figure 26. Graph of data bit error probability bound versus encoded symbol error probability for orthogonal convolutional codes of constraint length K = 7.

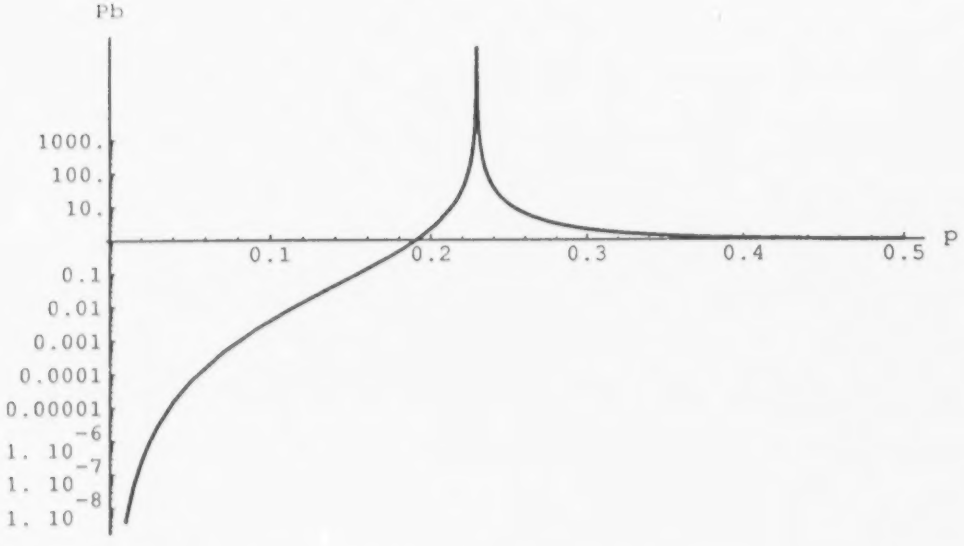


Figure 27. Graph of data bit error probability bound versus encoded symbol error probability for orthogonal convolutional codes of constraint length K = 3.

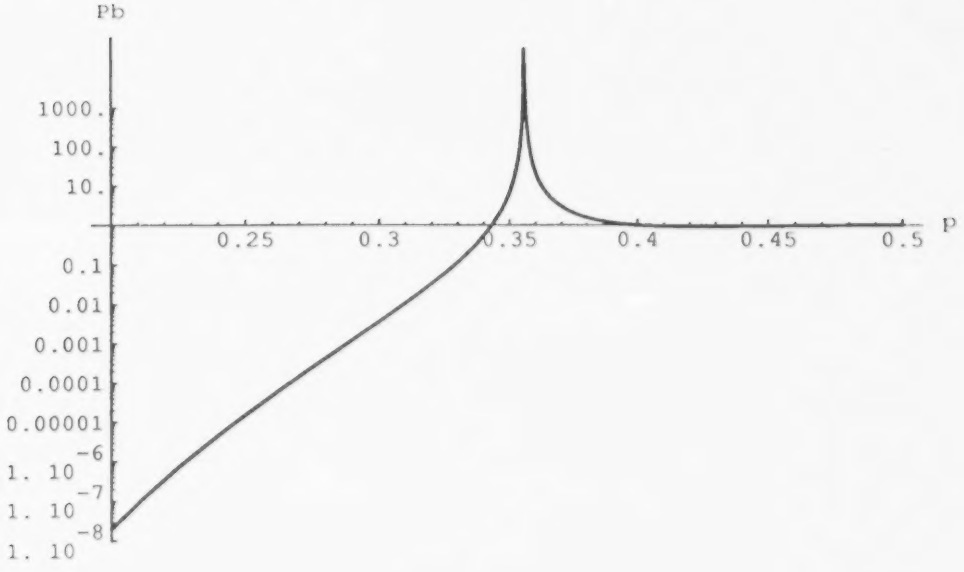


Figure 28. Graph of data bit error probability bound versus encoded symbol error probability for orthogonal convolutional codes of constraint length K = 5.

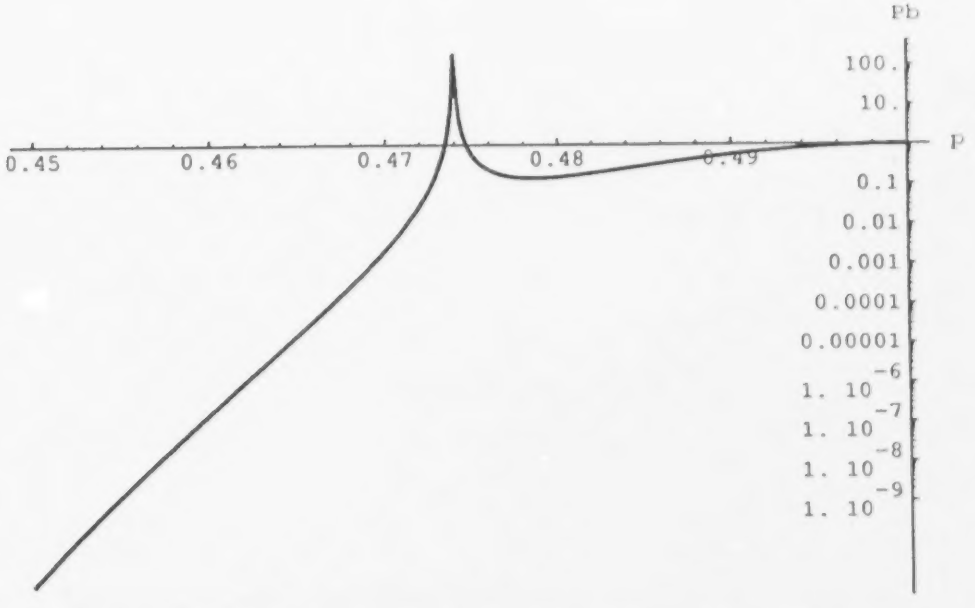


Figure 29. Graph of data bit error probability bound versus encoded symbol error probability for orthogonal convolutional codes of constraint length K = 10.

From observation of these curves, an estimate of the useful range of the bound can be made. If the assumptions are made that P_b should monotonically approach 0.5 as p approaches 0.5 and that the second derivative of the curve should always be less than zero as on typical bit error rate curves, then we can assume that the curve is no longer representative of the actual BER value when the steepness of the curve increases. Thus, for K = 10, K = 7, K = 5, and K = 3, the maximum values of p for which the bounds are valid are approximately 0.47, 0.41, 0.31, and 0.14 respectively.

Table 5 shows the symbol error rates and bit error rates obtained from simulations of the SSMA system using a code of constraint length K = 5 under a system load of 30 users for 10 different channel profiles. Also shown are the bit error rate upper bounds obtained from substituting the symbol error rate value into equation (2.2.3.11). These results show that when the value of p is below the validity cut-off value of 0.31, the bound gives reasonable estimates of the BER when compared to the simulation BER results. For values of p greater than 0.31, the bound becomes meaningless. The bound does not take into account the truncated path memory of the Viterbi decoder which causes coding gain losses on the order of a few tenths of a decibel [13]. This results in bound estimates that are lower than true values, particularly at very low BERs where the slope of the graph of the bound is very steep.

Channel	Simulation Symbol Error Rate, p	Simulation BER	BER Bound P_b
1	0.267	1.72×10^{-4}	1.08×10^{-4}
2	0.282	1.22×10^{-3}	5.77×10^{-4}
3	0.254	1.71×10^{-5}	2.52×10^{-5}
4	0.275	1.49×10^{-4}	2.55×10^{-4}
5	0.312	8.52×10^{-4}	1.31×10^{-2}
6	0.263	2.60×10^{-5}	6.75×10^{-5}
7	0.285	8.22×10^{-4}	7.59×10^{-4}
8	0.336	1.19×10^{-3}	2.97×10^{-1}
9	0.313	4.75×10^{-3}	1.55×10^{-2}
10	0.312	1.55×10^{-3}	1.28×10^{-2}
Average over valid values of p	0.271	4.01×10^{-4}	2.99×10^{-4}

Table 5. Comparison of simulation symbol error rate, bit error rate, and bit error rate bound for SSMA system using 1/32 code with 30 users.

Table 6 compares the average values of P_b from the error bounding method and the average bit error probabilities determined from full simulations of some SSMA systems. All values are from those simulations with values of p which are in the valid regions of the bounds. From these results, it seems reasonable to conclude that the average BER estimations using the encoded symbol error probabilities and the upper bit error probability bound are fairly accurate.

Code Rate	Number of System Users	Simulation Average BER	Average BER Bound, P_b
1/128	30	1.54×10^{-4}	1.74×10^{-4}
1/32	30	4.01×10^{-4}	2.99×10^{-4}
1/8	15	6.00×10^{-4}	1.03×10^{-3}

Table 6. Comparison of simulation bit error rates and bit error rate bounds for some SSMA systems.

Table 7 shows the results from the error bounding method for the SSMA system using different codes and under different loading conditions. These results can be used to extend the range of values obtained from the full simulations to much lower bit error rates.

The results of the channel analysis method for the uncoded SSMA system are shown in Table 8. Also included, where they are available, are the results from the full simulation method. The comparisons between the two sets of results show that fairly accurate estimates can be made in this way.

By extending the BER range, an estimate of the SSMA system capacity can be determined for an average BER of 1×10^{-4}. For systems using code rates of 1, 1/8, 1/32, 1/128, and 1/1024, the capacities are 9, 12, 24, 29, and 18 respectively.

Code Rate	Number of System Users	Average P_b
1/1024	25	3.31×10^{-2}
1/1024	20	1.48×10^{-2}
1/1024	15	1.02×10^{-7}
1/128	30	1.74×10^{-4}
1/128	25	5.30×10^{-6}
1/128	20	1.34×10^{-7}
1/32	30	2.99×10^{-4}
1/32	25	2.85×10^{-4}
1/32	20	2.47×10^{-6}
1/8	15	1.03×10^{-3}
1/8	12	1.29×10^{-4}
1/8	9	6.50×10^{-7}

Table 7. Average BERs for different SSMA systems determined with the error bound method.

Number of System Users	Average BER Channel Analysis Method	Average BER Full Simulation Method
12	3.00×10^{-4}	6.30×10^{-4}
9	4.77×10^{-5}	1.09×10^{-4}
6	6.84×10^{-7}	-
3	3.36×10^{-11}	-

Table 8. Average BER values from the channel analysis method and full simulation method for the uncoded SSMA system.

5.3 Discussion of Results

The results from the full simulation method, the error bounding method, and the channel analysis method are summarized in Figure 30. The system using the orthogonal convolutional code of rate 1/128 offers the best performance, followed by the 1/32, 1/8, 1, and 1/1024 code rates for most system capacities tested. Normally, we would expect that the systems using the lower code rates would perform best. However, the very high interference power levels encountered in low code rate SSMA systems limits the effectiveness of coding.

This is especially evident in the performance curve of the 1/1024 code rate SSMA system. At capacities above 20, it has the worst performance out of all the systems. At lower capacities where the interference power is lower, the coding gain is able to overcome the interference and system performance improves dramatically. At a capacity of approximately 18, the 1/1024 code rate SSMA system becomes superior to the 1/8 code rate system and the uncoded system. This result conforms to the behavior expected of the operation of FEC codes in very noisy channels.

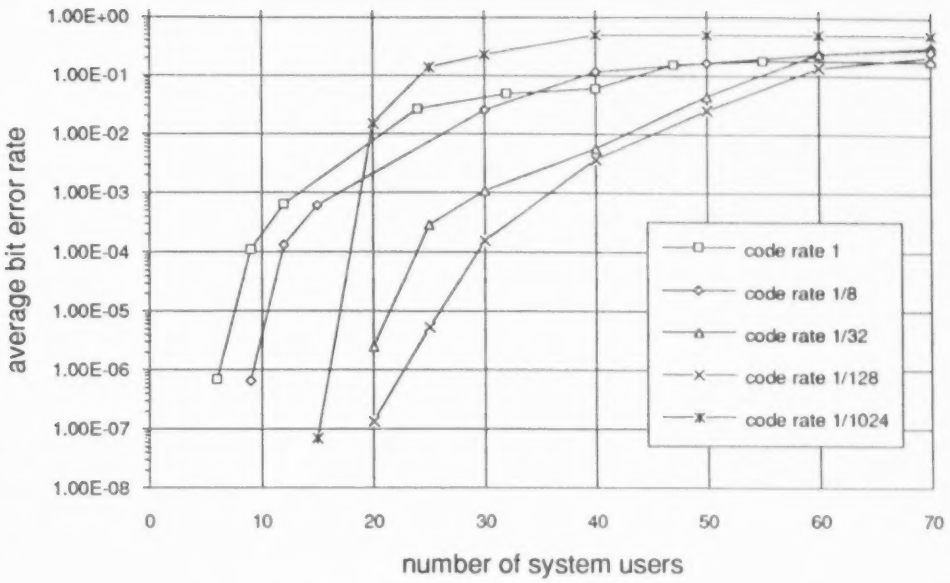


Figure 30. Summary of performance of the SSMA system using different code rates.

The system capacities determined here are lower than those determined in [9]. Both studies use similar spread spectrum to data bandwidth expansions. The study in [9] uses coherent binary phase shift keying (BPSK) instead of DPSK but does not account for any voice activation. The first factor could double the capacity of the system, but the second factor will reduce it by half again, therefore making the two studies comparable. However, the investigation in [9] used delay spreads of 100 ns and 250 ns on Rayleigh fading channels instead of 500 ns on experimentally based channels as used in this study. The comparison is made between the systems used in this study and the systems in [9] which use one antenna with selection and multipath diversity, spreading sequences of length 511, and (15,7) BCH codes or (7,4) Hamming codes.

At a BER of 10^{-4}, the 1/128 code rate SSMA system analyzed in this thesis was determined to be able to support approximately 29 users. The study

from [9] determined system capacities of approximately 60 when using the (15,7) BCH codes and 40 when using the (7,4) Hamming codes for channels with 100 ns delay spreads. For 250 ns delay spread channels, these systems capacities fell to approximately 50 and 40 respectively. Note that the multipath diversities are of orders 3 for the (7,4) code and 4 for the (15,7) code when the delay spread is set at 100 ns. For the 250 ns delay spread cases, the multipath diversities are of orders 8 for the (7,4) code and 9 for the (15,7) code. Third order multipath diversity is used in the system studied in this thesis.

The advantage of high multipath diversity orders is limited when using the SIRCIM channels because of the relatively low signal power available in the tails of the channel profiles which makes it mostly unusable by the RAKE receiver. From observation, most of the useful power is concentrated in the under 90 ns delay region of the profile. Even though the power in the delay region > 90 ns is low, it contributes non-negligible interference because it is spread over a fairly long duration of 410 ns.

The interference model used in this thesis is more severe than that used in [9]. The interference model in [9] was also based on a Gaussian distribution, but cross-correlation behavior of random spreading sequences instead of Gold sequences was assumed. If the same interference model was used in this thesis, the received signal to interference ratio would have been changed from (4.2.10) to (5.3.1) [9].

$$SIR = \frac{2}{(j-1)(\frac{2}{3N})} \qquad (5.3.1)$$

where N is the length of the partial or complete PN sequence

This would result in a decrease in noise power by factors of approximately 30, 11, 5, 3, and 1 for systems using code rates of 1, 1/8, 1/32, 1/128, and

1/1024 respectively. In such a case, superior system capacities would have resulted. However, the assumption that Gold sequences behave like random sequences is not valid and artificially inflates the capacity, especially for systems using long PN sequences.

The use of the error bounding method proved to be extremely helpful in analyzing the SSMA systems with BER values < 10^{-4}. Using the full simulation technique on such systems would require a tremendous amount of computation power and time. The greatest benefits arose when examining the 1/128 and 1/32 code rate systems since these systems usually offered the best performance at any given system capacity. Also, since the code rates are very low, these systems usually operate with a very high encoded symbol error rate. This makes estimating the symbol error rate required to compute the error bound an extremely quick procedure. The benefits of using the error bounding method are reduced when analyzing the 1/8 code rate system and are non-existent when analyzing the uncoded system.

Estimates of the 1/1024 code rate system were performed, but the results are not as reliable as for the other systems. This is because the error bound curve has a very steep bit error rate versus symbol error rate (P_b vs. p) slope. This makes any inaccuracies in estimating the symbol error rate very significant. For instance, if the actual value of p is 0.465 and the estimated value is 0.455 (a 2% error), then the error in P_b is of the order 10^4. For comparison, a 2% error in the estimate of p in a 1/128 code rate system results in an error of the order 10 in P_b.

The channel analysis method provided very quick estimates of the bit error rates for the uncoded SSMA system. It should not be used to estimate the value of the encoded symbol error rate (from which the bit error rate can then be bounded) in coded SSMA systems because any inaccuracies in this estimation

can cause significant variations in the bit error rate. For instance, the channel analysis method provided average bit error rates which were within a factor of approximately 2 of the full simulation results. This 50% error in the estimate of the encoded symbol error rate of a 1/128 code rate system would have caused a huge error in the BER.

6. Conclusions

There are three main areas in which results presented in this thesis were gathered:

a) multi-user interference modelling

b) simulation methodology

c) indoor radio SSMA system performance

Some suggestions concerning possible future work on SSMA for indoor wireless applications are also included at the end of the chapter.

6.1 Multi-user Interference Modelling

The assumption that multi-user interference can be modelled as white Gaussian noise was verified to be reasonable by viewing the frequency spectrum of the despread interference and performing the Kolmogorov-Smirnov test on interference data points. The mean normalized cross-correlations between partial and complete Gold sequences were also found in order to estimate the received signal to interference ratio.

6.2 Simulation Methodology

Three methods for estimating the mean bit error rate of the SSMA systems were used. The first was the full simulation method. This involved creating a computer model of the entire system and running simulations until enough bit errors were detected to provide a reasonable estimate of the bit error rate. The second was the error bounding method. This involved creating a computer model which would count the number of encoded symbol errors. Once an estimate of the symbol error probability was determined, it was inserted into the hard-decision orthogonal convolutional code error bound equation to get an approximation on the bit error rate.

The third method was the channel analysis method. The channel profiles each were examined to estimate the signal to interference ratio for the three stages of the RAKE receiver. The RAKE receiver was then analyzed as a three-channel DPSK receiver where each channel had a signal to noise ratio equivalent to the signal to interference ratio of its corresponding RAKE receiver stage.

The full simulation method is valid for all system bit error rates, but was extremely slow and required a tremendous amount of computing power when analyzing systems with low bit error rates. It was therefore used only for systems with bit error rates $\geq 10^{-4}$.

The error bounding method is not valid for high bit error rates because the bound is very loose for high encoded symbol error probabilities. However, it provided fairly accurate results for bit error rates $< 10^{-4}$. It proved to be extremely useful for systems using very low rate codes at very low bit error rates. These results could not have been gathered using the full simulation method because of the computing power required.

The channel analysis method is valid for analyzing the uncoded SSMA system. Almost instantaneous BER estimates could be made for any channel profile and interference level. The accuracy of this method was quite good also, providing BERs within a factor of 2 of the full simulation method results.

6.3 Indoor Radio SSMA System Performance

This thesis showed that the use of some very low rate orthogonal convolutional codes can significantly improve the performance of an indoor radio SSMA system when compared to uncoded systems. In particular, the system using the 1/128 code rate exhibited the best performance under most system loads.

For a mean bit error rate of 3×10^{-2}, the SSMA systems ranked from best to worst were 1/128, 1/32, 1/8, 1, and 1/1024. For mean bit error rates of 1×10^{-3} and 1×10^{-4}, the systems were ranked 1/128, 1/32, 1/1024, 1/8, and 1. The 1/1024 code rate system performed poorly at high system loads because of the extremely high interference power which negated the effect of the coding. At lower system loads, the power of its very low rate code allowed the system performance to quickly surpass that of some of the other systems which had previously exhibited superior performance. This shows that in order to get optimum performance, there must be a balance between increased coding gain from using lower rate codes and increased interference levels resulting from shorter partial PN sequences.

This thesis also showed that inherent multipath diversity in indoor digital radio systems may be limited because the signal power at long delays is generally too low to be useful. For the soft partitioned indoor office channels used in this thesis, a diversity order of 3 was used as it was observed that most of the channel profile components of significant power were located in the under 90 ns delay region.

The spectral efficiency of the system examined in this thesis turned out to be quite low, even when using the 1/128 code rate. This may be the result of the asynchronous nature of the system. The fact that it used DPSK reduced its capacity by less than 2 when compared to BPSK, and the unsynchronized transmissions from mobile terminals did not allow for optimum interference suppression.

6.4 Future Work

Since SIRCIM can also produce hard partitioned office environment and warehouse environment channels, it may also be interesting to examine the

effect that different environments have on SSMA system performance. These different channels may permit higher orders of multipath diversity.

Because of the relatively low spectral efficiencies of all the SSMA systems analyzed, it would also be interesting to determine the effect of synchronizing the system. By time synchronizing the transmissions, better PN sequences may be used. A possible alternative to using Gold sequences is using Walsh functions. All Walsh functions are orthogonal to one another, thus they could virtually eliminate the problem of multi-user interference. However, their orthogonality depends on the codes being time synchronized. A study could be performed to analyze the effect of the indoor multipath fading channel on a system employing Walsh functions. Such a system has been implemented in a field trial for outdoor cellular telephone purposes [14].

Of course, the possible improvements just listed would make the system much more complicated and perhaps, more limited. As well, additional overhead would be required for synchronization. The cost, complexity, and efficiency of such systems would have to be compared to other systems to determine which one best fits the needs of the end user. After all, personal communications should be designed to suit the person!

Bibliography

[1] K.S. Gilhousen, *et al*, "On the Capacity of a Cellular CDMA System," *IEEE Transactions on Vehicular Technology*, vol.40, no.1, pp. 303-312, May 1991.

[2] A.J. Viterbi, "Spread Spectrum Communications - Myths and Realities," *IEEE Communications Magazine*, vol.17, pp.11-18, May 1979.

[3] G.D. Boudreau, D.D. Falconer, and S.A. Mahmoud, "A Comparison of Trellis Coded Versus Convolutional Coded Spread-Spectrum Multiple Access Systems," *IEEE Journal on Selected Areas in Communications*, vol.8, no.4, pp. 628-639, May 1990.

[4] A.J. Viterbi, "Very Low Rate Convolutional Codes for Maximum Theoretical Performance of Spread-Spectrum Multiple Access Channels," *IEEE Journal on Selected Areas in Communications*, vol.8, no.4, pp. 641-649, May 1990.

[5] Block Oriented Systems Simulator (BOSS), Version 2.7, Comdisco Systems, 1991.

[6] T.S. Rappaport and S.Y. Seidel, "SIRCIM Simulation of Indoor Radio Channel Impulse response Models," VTIP, 1990.

[7] T.S. Rappaport, S.Y. Seidel and K. Takamizawa, "Statistical Channel Impulse Response Models for Factory and Open Plan Building Radio Communication System Design," *IEEE Transactions on Communications*, vol.39, no.5, pp. 794-807, May 1991.

[8] Al Javed, "Digital Cellular Technology and Performance", *Proceedings of Wireless 92*, Calgary, Alberta, July 1992.

[9] M. Kavehrad and P.J. McLane, "Spread Spectrum for Indoor Digital Radio", *IEEE Communications Magazine*, vol.25, no.6, pp. 32-40, June 1987.

[10] D.L. Schilling *et al*, "Spread Spectrum for Commercial Communications," *IEEE Communications Magazine*, vol.29, no.4, pp. 66-79, April 1991.

[11] George L. Turin, "Introduction to Spread-Spectrum Antimultipath Techniques and Their Application to Urban Digital Radio," *Proceedings of the IEEE*, vol.68, no.3, pp. 328-352, March 1980.

[12] P.T. Brady, "A Statistical Analysis of On-Off Patterns in 16

Conversations," *Bell System Technical Journal*, vol.47, pp. 73-91, January 1968.

[13] J.G. Proakis, *Digital Communications 2nd Edition*, McGraw-Hill, New York, 1989.

[14] A. Salmasi and K.S. Gilhousen, "On the System Design Aspects of Code Division Multiple Access (CDMA) Applied to Digital Cellular and Personal Communications Networks," *41st IEEE Vehicular Technology Conference*, St. Louis, MO, pp. 57-62, May 1991.

[15] Robert C. Dixon, *Spread Spectrum Systems 2nd ed.*, John Wiley and Sons, New York, 1984.

[16] Wesley W. Peterson, *Error Correcting Codes*, MIT Press, Cambridge Massachusetts, 1961.

[17] Bernard Sklar, *Digital Communications Fundamentals and Applications*, Prentice Hall, New Jersey, 1988.

[18] A.J. Viterbi and J.K. Omura, *Principles of Digital Communication and Coding*, McGraw-Hill, New York, 1979.

[19] R.R. Green, "A Serial Orthogonal Decoder," *JPL Space Programs Summary*, vol.37-39-IV, pp. 247-253, Jet Prop. Lab., Pasadena, CA, 1966.

[20] Kamilo Feher, *Advanced Digital Communications*, Prentice-Hall, Inc., New Jersey, 1987.

[21] William C.Y. Lee, *Mobile Communications Engineering*, McGraw-Hill, New York, 1982.

[22] K.S. Gilhousen and R. Padovani, "Increased Capacity Using CDMA for Mobile Satellite Communication," *IEEE Journal on Selected Areas in Communications*, vol.8, no.4, pp. 503-514, May 1990.

[23] William C.Y. Lee, "Overview of Cellular CDMA," *IEEE Transactions on Vehicular Technology*, vol.40, no.2, pp. 291-302, May 1991.

[24] M. Kavehrad and Bhaskar Ramamurthi, "Direct-Sequence Spread Spectrum with DPSK Modulation and Diversity for Indoor Wireless Communications," *IEEE Transactions on Communications*, vol.COM-35, no.2, pp. 224-236, February 1987.

[25] K. Pahlavan and M. Chase, "Performance of Code-Division Multiple-Access Orthogonal Codes for Indoor Radio Communications," *IEEE Transactions on Communications*, vol.38, no.5, pp. 574-577, May 1990.

[26] E. Geraniotis, "Direct-Sequence Spread-Spectrum Multiple-Access Communications Over Nonselective and Frequency-Selective Rician Fading Channels," *IEEE Transactions on Communications*, vol.COM-34, no.8, pp. 756-764, August 1986.

[27] Paul D. Shaft, "Low-Rate Convolutional Code Applications in Spread-Spectrum Communications," *IEEE Transactions on Communications*, vol.COM-25, no.8, pp. 815-821, August 1977.

[28] M. Kavehrad and P.J. McLane, "Performance of Direct Sequence Spread Spectrum for Indoor Wireless Digital Communication," *Proceedings of GLOBECOM '85*, New Orleans, LA, Nov. 1985.

[29] T.S. Rappaport and S.Y. Seidel, "Multipath Propagation Models for In-Building Communications," *Fifth International Conference on Mobile Radio and Personal Communications*, Coventry, England, Dec. 1989.

[30] D.A. Hawbaker and T.S. Rappaport, "Indoor Wideband Radiowave Propagation Measurements at 1.3 GHz and 4.0 GHz," *Electronics Letters*, vol.26, no.21, pp. 1800-1802, 1990.

[31] R. Price and P.E. Green, Jr., "A Communication Technique for Multipath Channels," *Proceedings of the IRE*, vol.46, pp. 555-570, March 1958.

[32] Michel C. Jeruchim, "Techniques for Estimating the Bit Error Rate in the Simulation of Digital Communication Systems," *IEEE Journal on Selected Areas in Communications*, vol.SAC-2, no.1, pp. 153-170, Jan. 1984.

[33] R.L. Winkler and W.L. Hays, *Statistics, Probability, Inference, and Decision, 2nd Edition*, Holt, Rinehart, and Winston, Inc., 1975.

[34] I.N. Gibra, *Probability and Statistical Inference for Scientists and Engineers*, Prentice-Hall, Inc., New Jersey, 1973.

[35] William H. Press, *et al*, *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, Cambridge, Massachusetts, 1986.

[36] Michel K. Ochi, *Applied Probability & Stochastic Processes*, John Wiley and Sons, Inc., New York, 1990.

Appendix 1

Simulation Systems and Modules

A1.i Encoder

Figure A1 shows OC_ENCODER (PART1) which is the component building block for an orthogonal convolutional encoder. The top sample-and-hold module represents one stage of the linear convolutional shift register. The variable delay, the select switch, and the X-OR gate are components of one stage of an Hadamard encoder. A K^{th} order orthogonal convolutional encoder that performs identically to the one shown in Figure 6 can be formed by making the modules OC_ENCODER (PART2) and OC_ENCODER (ARBITRARY STAGES) shown in Figures A2 and A3. Three parameters must be specified for OC_ENCODER (ARBITRARY STAGES): the original uncoded input data rate, the constraint length K, and the position of each particular block in the linking sequence (ie. the N^{th} component of the K component encoder). This last parameter is written for each OC_ENCODER (PART2) module by initialization code. The initialization code for this module is given as ROUTINE_ENCODER_INIT.

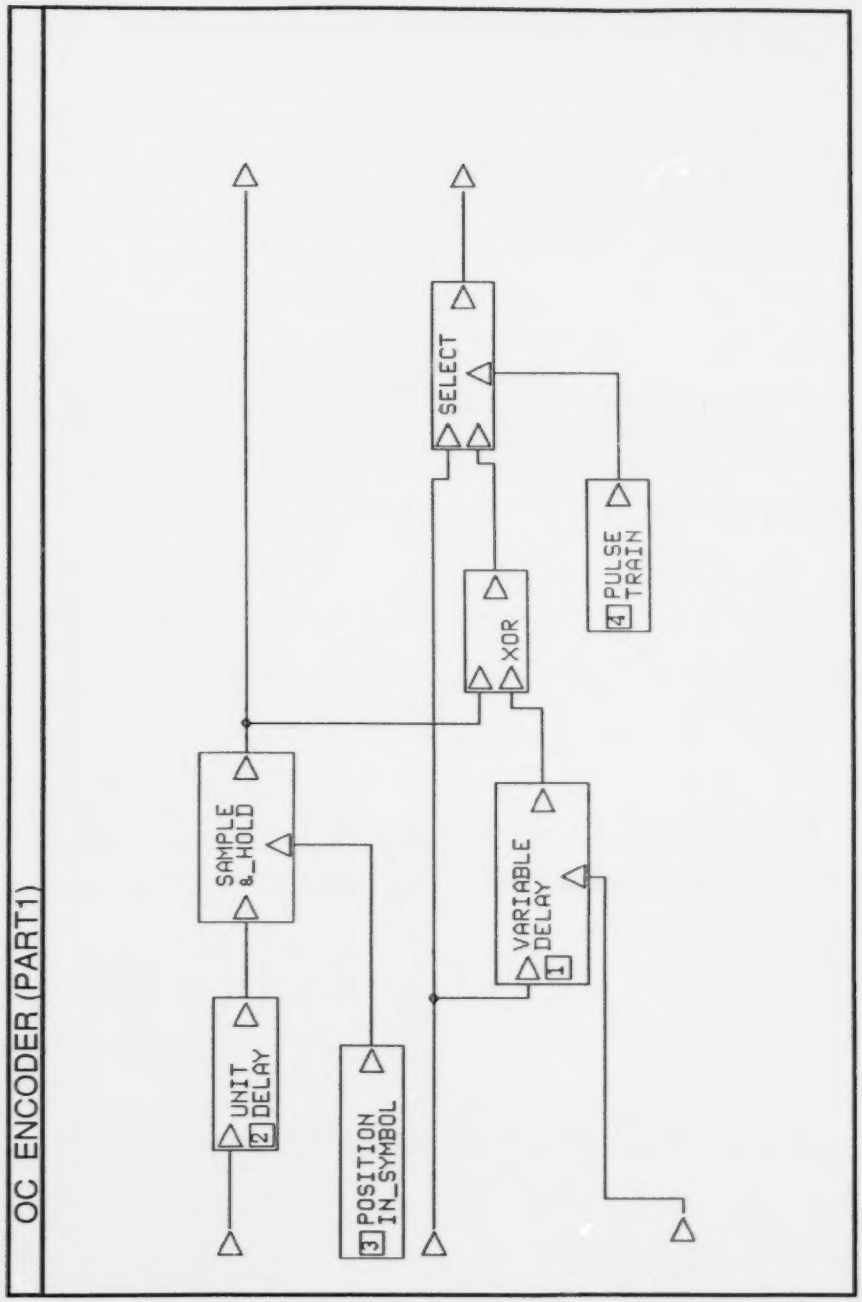


Figure A1. One stage of an orthogonal convolutional encoder.

MODULE NAME: OC_ENCODER (PART1)
GROUP: SOURCE ENCODERS/DECODERS
DATABASE: /home/suns/rtse/model_db/
AUTHOR: rtse
CREATION DATE: 5-Feb-1992 14:09:26

DESCRIPTION:

Module for Nth stage of orthogonal convolutional encoder.

REVISIONS:

Author : rtse
Date : 5-Feb-1992 14:09:26
Description:

5-Feb-1992 14:09:49
Module CREATION.

INPUT SIGNALS:

DELAY NUM Type: INTEGER
Lower Limit: 1
Upper Limit: 2500

The number of sample delays required for this stage of the orthogonal convolutional encoder. This delay is calculated via:

delay = (2.0^(N-1))/(2^K * bit rate * dt)

O-ENCODER INPUT Type: LOGICAL
Lower Limit: NIL
Upper Limit: NIL

Input from the (N-1)th stage of the orthogonal encoder.

SR-INPUT Type: LOGICAL
Lower Limit: NIL
Upper Limit: NIL

Input from the (N-1)th stage of the shift register of the convolutional encoder.

OUTPUT SIGNALS:

O-ENCODER OUTPUT Type: LOGICAL
Lower Limit: NIL
Upper Limit: NIL

The output to the (N+1)th orthogonal encoder stage.

SR OUTPUT Type: LOGICAL
Lower Limit: NIL
Upper Limit: NIL

The output to the next shift register stage of the convolutional encoder.

PARAMETERS:

BIT RATE Type: REAL
Lower Limit: 3.0E-39
Upper Limit: 1.7E38

The input data bit rate.

N Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

This value should equal the stage within the orthogonal convolutional coder that this module is in (ie.it can equal from 1 to K for a K-stage coder).

K Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

fdbhfdg

MODULES USED IN BLOCK DIAGRAM:
XOR
PULSE TRAIN
SELECT

```
   SAMPLE & HOLD
   POSITION IN SYMBOL
   UNIT DELAY
   VARIABLE DELAY

PARAMETER VALUES FOR INSTANCES IN BLOCK DIAGRAM:

VARIABLE DELAY (key 1)
   MAX DELAY == 2500

UNIT DELAY (key 2)
   INITIAL VALUE == .FALSE.

POSITION IN SYMBOL (key 3)
   TIME DELAY TO INPUT (SEC) == 0
   SYMBOL FRAC FOR SAMPLE TIME == 1
   SYMBOL RATE (HZ) == $BIT RATE

PULSE TRAIN (key 4)
   PULSE WIDTH (SEC.) == 1.0 / ('BIT RATE' * 2.0^('K' - 'N' + 1.0))
   PULSE RATE (HZ.) == (2.0^('K' - 'N') * 'BIT RATE')

INITIALIZATION CODE:
   (none)
```

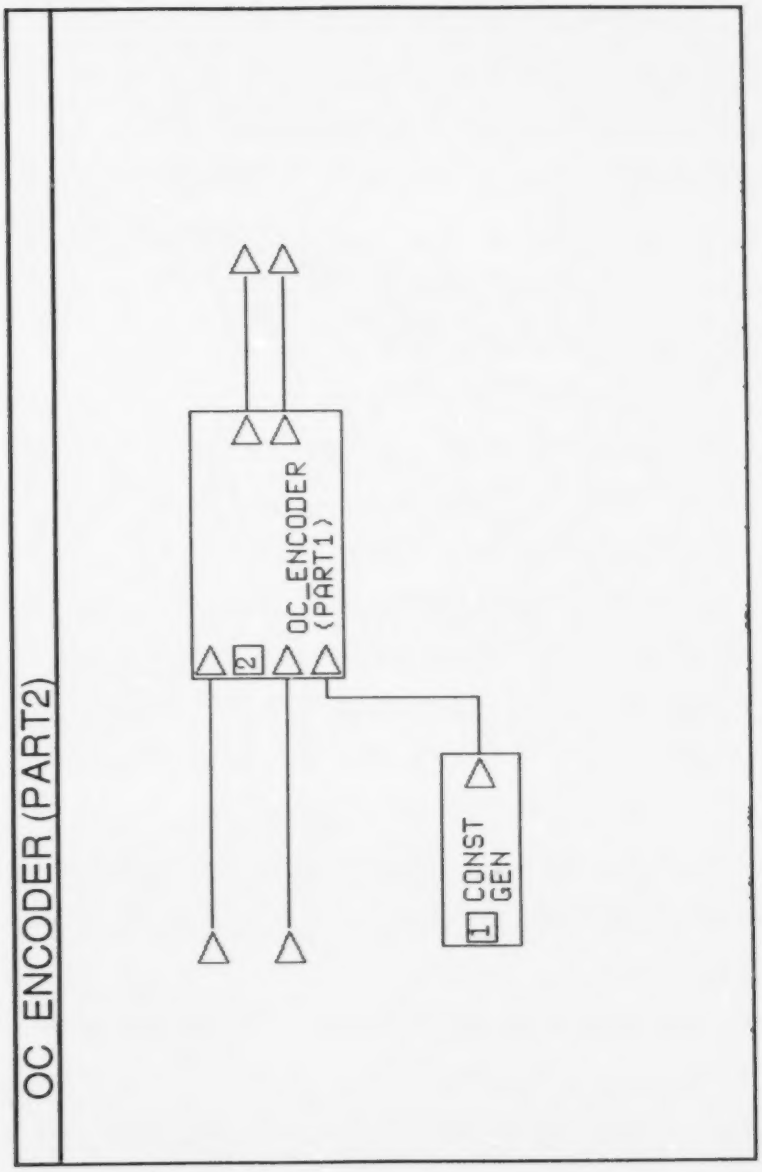


Figure A2. Intermediate stage for constructing a K-stage orthogonal convolutional encoder.

MODULE NAME: OC_ENCODER (PART2)
GROUP: SOURCE ENCODERS/DECODERS
DATABASE: /home/suns/rtse/model_db/
AUTHOR: rtse
CREATION DATE: 5-Feb-1992 14:20:58

DESCRIPTION:

This module is one stage of a K stage orthogonal convolutional encoder. This particular design is for use with the REPLICATION function of BOSS.

REVISIONS:

Author : rtse
Date : 5-Feb-1992 14:58:57
Description:

Edited 5-Feb-1992 14:58:57, No Edit Description Entered.

Author : rtse
Date : 5-Feb-1992 14:23:57
Description:

Edited 5-Feb-1992 14:23:57, No Edit Description Entered.

Author : rtse
Date : 5-Feb-1992 14:20:58
Description:

5-Feb-1992 14:21:18
Module CREATION.

INPUT SIGNALS:

CONNECT TO SHIFT REGISTER OF N-1 STAGE Type: LOGICAL
Lower Limit: NIL
Upper Limit: NIL

This input should be connected to the coded output of the previous stage. If this module is the first stage, then a .false. input should be connect here.

CONNECT TO OUTPUT OF STAGE N-1 Type: LOGICAL
Lower Limit: NIL
Upper Limit: NIL

This input should be connected to the shift register output of the previous stage. If this module is the first stage,

then the data to be encoded should be input here.

OUTPUT SIGNALS:

SHIFT REGISTER OUTPUT Type: LOGICAL
Lower Limit: NIL
Upper Limit: NIL

This is the coded output of the Nth stage coder.

CODED OUTPUT Type: LOGICAL
Lower Limit: NIL
Upper Limit: NIL

This is the output of the Nth shift register stage.

PARAMETERS:

DATA RATE Type: REAL
Lower Limit: 3.0E-39
Upper Limit: 1.7E38

This value is equal to the rate (in bits/sec) of the data to be encoded by the K stage orthogonal convolutional encoder.

K Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

The value K is the constraint length of the orthogonal convolutional encoder.

N Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

The value N corresponds to the stage of this module in a K stage orthogonal convolutional encoder.

MODULES USED IN BLOCK DIAGRAM:
OC_ENCODER (PART1)
CONST GEN

PARAMETER VALUES FOR INSTANCES IN BLOCK DIAGRAM:

CONST GEN (key 1)
CONSTANT VALUE == Round (2.0^('N' - 1.0) / ((2.0^'K' * 'DATA RATE') * 'DT'))

OC_ENCODER (PART1) (key 2)
BIT RATE == $DATA RATE
N == $N
K == $K

INITIALIZATION CODE:
(none)

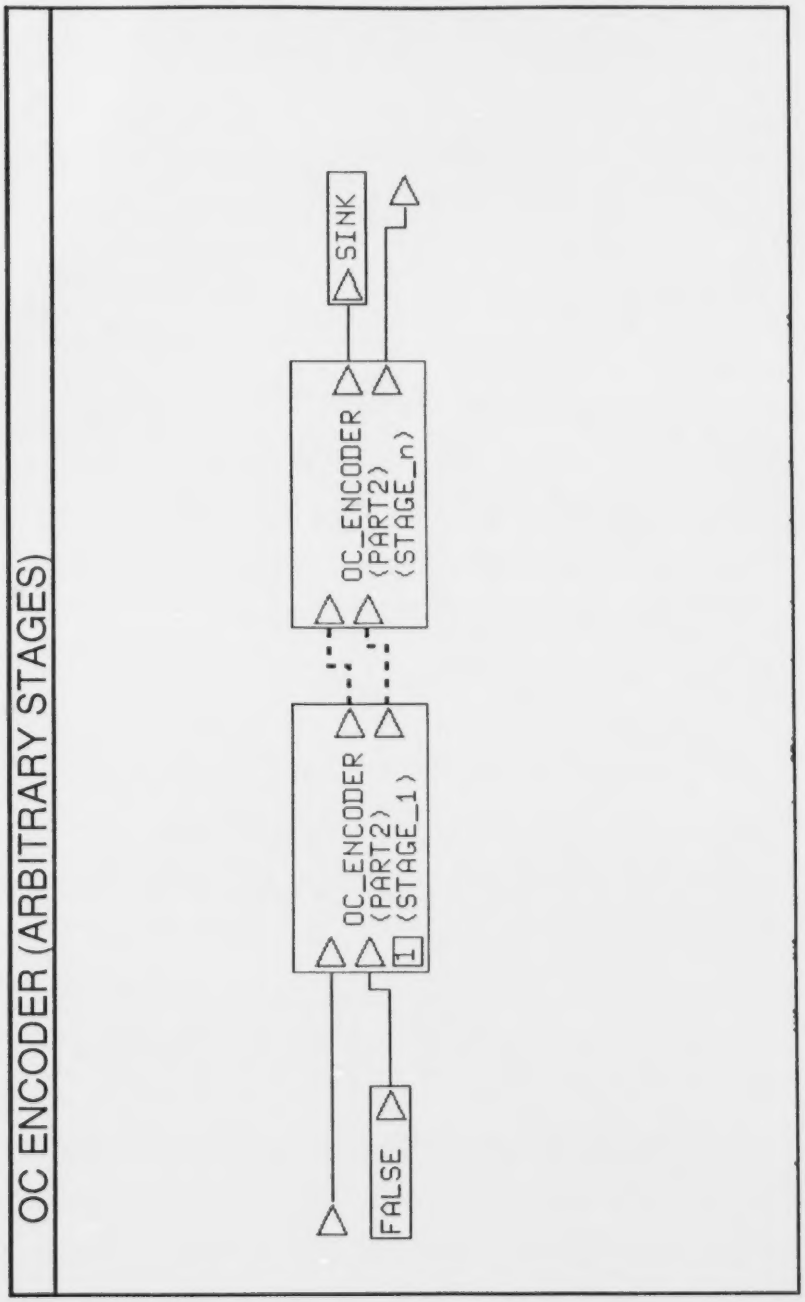


Figure A3. A K-stage orthogonal convolutional encoder.

MODULE NAME: OC ENCODER (ARBITRARY STAGES)
GROUP: SOURCE ENCODERS/DECODERS
DATABASE: /home/suns/rtse/model_db/
AUTHOR: rtse
CREATION DATE: 5-Feb-1992 14:32:38

DESCRIPTION:

This module is an orthogonal convolutional encoder with a maximium constraint length of 10.

REVISIONS:

Author : rtse
Date : 13-Feb-1992 11:47:32
Description:

Edited 13-Feb-1992 11:47:32, No Edit Description Entered.

Author : rtse
Date : 5-Feb-1992 15:00:07
Description:

Edited 5-Feb-1992 15:00:07, No Edit Description Entered.

Author : rtse
Date : 5-Feb-1992 14:32:38
Description:

5-Feb-1992 14:32:59
Module CREATION.

INPUT SIGNALS:

INPUT DATA Type: LOGICAL
Lower Limit: NIL
Upper Limit: NIL

The logical signal to be encoded with an orthogonal convolutional code.

OUTPUT SIGNALS:

ENCODED OUTPUT Type: LOGICAL
Lower Limit: NIL
Upper Limit: NIL

The coded output.

PARAMETERS:

DATA RATE Type: REAL
Lower Limit: 3.0E-39
Upper Limit: 1.7E38

This value is equal to the rate (in bits/sec) of the data to be encoded by the K stage orthogonal convolutional encoder.

CODE CONSTRAINT LENGTH Type: INTEGER
Lower Limit: 1
Upper Limit: 10

The constraint length of the orthogonal convolutional code. This value is limited to a maximum of 10.

COMPUTED PARAMETERS:

N Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647
Vector Length: (CODE CONSTRAINT LENGTH)

The value N corresponds to the stage of this module in a K stage orthogonal convolutional encoder.

MODULES USED IN BLOCK DIAGRAM:
FALSE
SINK
OC_ENCODER (PART2)

PARAMETER VALUES FOR INSTANCES IN BLOCK DIAGRAM:

OC_ENCODER (PART2) (key 1)
DATA RATE == $DATA RATE
K == $CODE CONSTRAINT LENGTH
N == $N

INITIALIZATION CODE:

Subroutine: zizOCENCODERARBITMPFNKDNK
Arguments:
CODE CONSTRAINT LENGTH

N

Description:

The initialization code writes the stage number to each module which makes up a stage within the encoder.

```
C                    ROUTINE_ENCODER_INIT

C     This initialization code is for the OC ENCODER (ARBITRARY STAGES)
C     module.  It writes the stage number to the OC_ENCODER (PART2)
C     submodules.

      subroutine code_init(k,n)

      integer k,n(12),i

      do i = 1,k,1
       n(i) = i
      enddo

      return
      end
```

A1.ii DPSK Modulator

Figure A4 shows the block diagram of the DPSK modulator. Note that this gives a base-band representation of DPSK modulation. The operation of binary DPSK can be described by:

$$\begin{aligned} \text{output}(t) &= -\text{output}(t-T) && \text{if input}(t) = 1 \\ &= +\text{output}(t-T) && \text{if input}(t) = -1 \end{aligned}$$

where T is the period of one digital bit and output(t<0) = 1

Because DPSK is to be non-coherently demodulated, the model allows for the user to specify an arbitrary phase offset which represents the difference between the carrier phase and the demodulating waveform phase (the demodulator described in A1.v. is assumed to operate at the reference phase offset of zero). This phase offset is executed by the final multiplication block within the module.

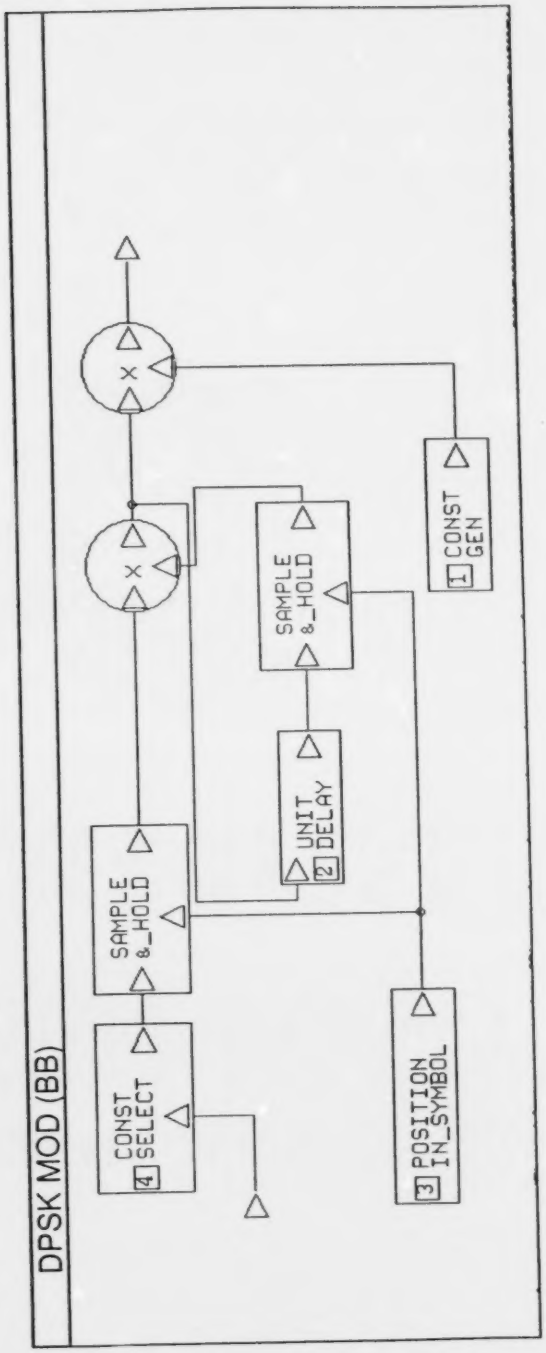


Figure A4. Baseband representation of a DPSK modulator.

MODULE NAME: DPSK MOD (BB)
GROUP: DIGITAL MODULATORS
DATABASE: /home/suns/rtse/model_db/
AUTHOR: rtse
CREATION DATE: 23-Sep-1991 14:22:25

DESCRIPTION:

This modulate simulates DPSK modulation at the base-band level. The output phase is shifted by PI radians if a +1 binary signal is transmitted. The phase is not shifted if a -1 binary signal is transmitted.

The parameter PHASE OFFSET represents an absolute phase offset in the carrier signal. However, since this is only a baseband representation, the phase offset of the carrier is shifted to the baseband signal instead.

REVISIONS:

Author : rtse
Date : 23-Sep-1991 22:19:37
Description:

Edited 23-Sep-1991 22:16:42, No Edit Description Entered.
Edited 23-Sep-1991 22:19:37, No Edit Description Entered.

Author : rtse
Date : 23-Sep-1991 14:22:25
Description:

23-Sep-1991 14:22:47
Module CREATION.

INPUT SIGNALS:

BINARY DATA INPUT Type: LOGICAL
Lower Limit: NIL
Upper Limit: NIL

The binary signal which is to be DPSK modulated.

OUTPUT SIGNALS:

DPSK MODULATED OUTPUT Type: COMPLEX

```
   Lower Limit: (-1.7E38 -1.7E38)
   Upper Limit: (1.7E38 1.7E38)

   The DSPK modulated signal corresponding to the input
   binary data.  The signal is modulated at the frequency

   --------

PARAMETERS:

   BAUD_RATE                                          Type: REAL
   Lower Limit: 3.0E-39
   Upper Limit: 1.7E38

   The rate (in bits/sec) of the input to this module.

   --------

   PHASE OFFSET                                       Type: REAL
   Lower Limit: -1.7E38
   Upper Limit: 1.7E38

   The absolute phase offset of the equivalent DPSK
   carrier waveform (in radians).

   --------

MODULES USED IN BLOCK DIAGRAM:
   CONST SELECT
   POSITION IN_SYMBOL
   UNIT DELAY
   SAMPLE & HOLD
   MULTIPLIER
   CONST GEN

PARAMETER VALUES FOR INSTANCES IN BLOCK DIAGRAM:

CONST GEN (key 1)
   CONSTANT VALUE ==  j* ( sin ('PHASE OFFSET')) +  cos ('PHASE OFFSET')

UNIT DELAY (key 2)
   INITIAL VALUE == (1.0 , 0.0)

POSITION IN_SYMBOL (key 3)
   TIME DELAY TO INPUT (SEC) == 0
   SYMBOL FRAC FOR SAMPLE TIME == 0
   SYMBOL RATE (HZ) == $BAUD_RATE

CONST SELECT (key 4)
   TRUE VALUE == (-1.0 , 0.0)
   FALSE VALUE == (1.0 , 0.0)

INITIALIZATION CODE:
   (none)
```

A1.iii Gold Sequences

The BOSS primitive and its code which produces the Gold sequence output used for spectral spreading and despreading of the FEC encoded symbols are given here. The block diagram of this module is shown in Figure A5 and consists only of one input and one output. The code is given as ROUTINE_GOLD_PN_PRIM. The input to the module should be connected to an impulse train of rate equal to the chip rate of the SSMA system. The output is the Gold sequence. The parameters required for the primitive are the Gold sequence length and the user number. The Gold sequence length value is used to allocate memory to storing the code. The user number parameter tells the primitive which Gold sequence to use out of the 2^n-1 available in the file 'gold_codes.dat'. The primitive will read the state of all 2^n-1 chips of the sequence upon initialization and, add one extra chip of random state at the end to create the modified Gold sequence, and store store the sequence in memory. It will output the chips at the appropriate rate during the simulation. The rate of the output is controlled by an external clock which sends an impulse train at the chip rate. The primitive will output one chip of the sequence every impulse and the sequence is repeated once it reaches the end.

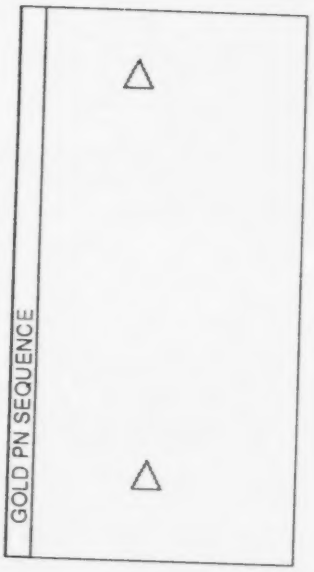


Figure A5. Primitive block diagram for the Gold sequence generator.

MODULE NAME: GOLD PN SEQUENCE
GROUP: DIGITAL SOURCES
DATABASE: /home/suns/rtse/model_db/
AUTHOR: rtse
CREATION DATE: 26-Aug-1992 17:16:55

DESCRIPTION:

A pseudo-noise code generator. These particular codes are Gold codes. Gold codes are used because they have an upper bound to their cross-correlation values.
The input parameters are the the size and the user number.
This module modifies the Gold code by adding one random bit to the end of the sequence. This is to make its length a power of 2.
The size parameter limits the array size for storing the gold sequence. This must be equal to the length of the modified sequence.
A timing signal which gives an impulse every chip period is required as an input to this module.
The module output is in complex form. (1,0) represents a positive bit and (-1,0) represents a negative bit. This facilitates the multiplication process between the PN code and the modulated waveform.

REVISIONS:

Author : rtse
Date : 27-Aug-1992 21:25:58
Description:

Edited 27-Aug-1992 21:25:58, No Edit Description Entered.

Author : rtse
Date : 26-Aug-1992 17:21:14
Description:

Edited 26-Aug-1992 17:21:14, No Edit Description Entered.

Author : rtse
Date : 26-Aug-1992 17:16:55
Description:

26-Aug-1992 17:17:00
Module CREATION.

INPUT SIGNALS:

CHIP TIMING SIGNAL Type: LOGICAL
Lower Limit: NIL
Upper Limit: NIL

When true, this signal indicates that the next chip in the sequence should be output.

OUTPUT SIGNALS:

GOLD OUT Type: COMPLEX
Lower Limit: (-1.0 -1.0)
Upper Limit: (1.0 1.0)

This output is a complex signal. A (1,0) represents a positive bit, and a (-1,0) represents a negative bit. This complex format was chosen over a logical format because the pn sequence is usually used to multiply a complex modulated signal. The given format will negate the need for a logical to numeric conversion module.

PARAMETERS:

USER NUMBER Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

The user number corresponds to the nth user in the multiple access system.

SIZE Type: INTEGER
Lower Limit: 1
Upper Limit: 5000

This parameter determines the memory allocated to store the gold sequence. Each gold sequence is of size (2^K) where K is an integer. For maximum memory usage efficiency, the parameter SIZE should be equal to or slightly greater than the value on the first line of the file 'gold_codes' which this module reads its gold sequences from.

MODULES USED IN BLOCK DIAGRAM:

(none - This module is a BOSS Primitive)

INITIALIZATION CODE:
(none)

```
C                    ROUTINE GOLD_PN_PRIM

      subroutine goldpn(out_pn,size,chip_sig,user_no,chip_no,seq_len,
     &                  code)


      implicit none
      integer user_no, seq_len, c, garbage, chip_no, code(*),lun
      integer size, i,seed,uni
      complex out_pn
      logical*1 chip_sig

      include '/bossdir/system/BOSSFORTRAN.INC'

C     Read the correct pn_code into memory locations code(x) and add one
C     extra chip to modify the Gold sequence length to a power of 2.
      if (curtime.lt.dt) then
        chip_no = 0
        call lib$get_lun(lun)
        open(unit=lun,file='gold_codes.dat',status='old')
        read(lun,1000) seq_len
        read(lun,1000) garbage
 1000   format(i5)
        do c = 1,((user_no-1)*(seq_len+1)),1
          read(lun,1002) garbage
 1002     format(i2)
        enddo
        do c = 1,seq_len,1
          read(lun,1002) code(c)
        enddo
        close(lun,status='keep')
        call lib$free_lun(lun)
        call lib$get_lun(lun)
        open(unit=lun,file='seed.dat',status='old')
        read(lun,*) seed
        close(lun)
        call lib$free_lun(lun)
        i = seq_len+1
        code(i) = uni(seed)
        call lib$get_lun(lun)
        open(unit=lun,file='seed.dat',status='old')
        write(lun,*) seed
        close(lun)
      endif


C     Whenever a chip period is up, the output will change accordingly.
      if (chip_sig) then
        chip_no = chip_no + 1
        if (chip_no.gt.seq_len+1) then
           chip_no = 1
        endif
      endif

      if (code(chip_no).eq.1) then
         out_pn = (1.0,0.0)
      else
         out_pn = (-1.0,0.0)
      endif
```

```
return
end

integer function uni(seed)
 integer seed
 real ran,x
 x = ran(seed)
 if (x.gt.0.5) then
   uni=1
 else
   uni=-1
 endif
return
end
```

A1.iv Multipath Channel

Multipath channel profiles generated by SIRCIM (as described in Section 3.1) are read in by the BOSS primitive given here. The module reads in a profile from the data file 'fc*xx*a' where *xx* corresponds to a 1 or 2 digit number from 1 to 50 inclusive which is set by the user. Another parameter allows the user to specify which channel profile within the designated file he wishes to begin at (each file contains 19 channel profiles).

Figure A6 shows the module constructed to perform the time domain convolution between the complex input signal and the channel impulse response. The convolution between the channel impulse response and the input signal is accomplished transforming both the input and the channel impulse response into vectors and using a dot product between the two. The result is the representation of one users' transmitted signal after multipath channel propagation.

The module IMPULSE RESPONSES FROM FILE is a primitive which reads the channel profile and then transforms it into a vector which is used in the dot product. Its block diagram is given in Figure A7 and its code is given as ROUTINE_IMPULSE_RESPONSE_FROM_FILE_PRIM.

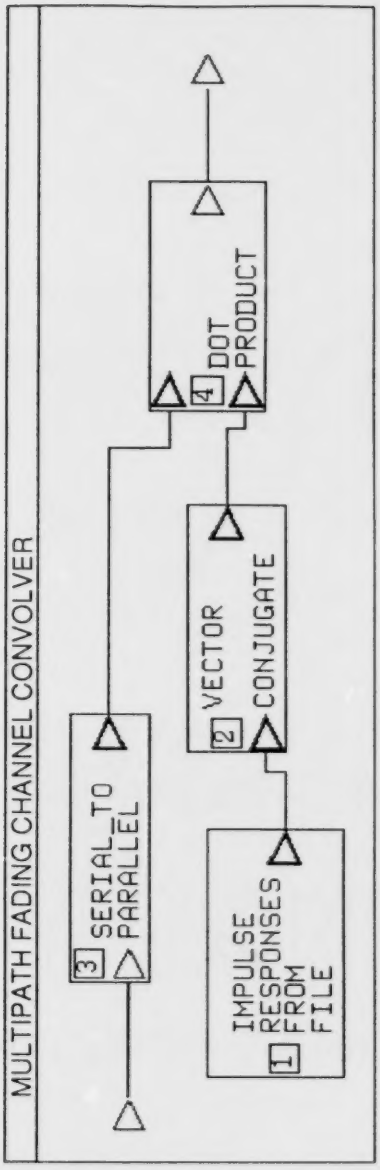


Figure A6. Multipath fading channel convolver.

MODULE NAME: MULTIPATH FADING CHANNEL CONVOLVER
GROUP: CHANNELS
DATABASE: /home/suns/rtse/model_db/
AUTHOR: rtse
CREATION DATE: 21-Jul-1991 22:17:38

DESCRIPTION:

This module convolves a complex input signal with the impulse response of a fading multipath channel (which is read from the file 'fc#a' where # corresponds to the one or two digit parameter CHANNEL NUMBER).

The sampling period must correspond to the sampling period of the multipath channel's impulse response.

~~Whenever the channel impulse response is updated, the output 'DELAY' is recalculated to give the number of sample delays are between the start of the channel impulse response and the impulse of maximum magnitude within that channel impulse response.~~

The parameter 'CHANNEL START NUMBER' determines which set of impulse responses from the channel impulse response file is the initial one.

The parameter 'CHANNEL START NUMBER' determines which set of impulse responses from the channel impulse response file to start at.

REVISIONS:

Author : rtse
Date : 17-Jan-1992 11:28:54
Description:

Edited 17-Jan-1992 11:28:54, No Edit Description Entered.

Author : rtse
Date : 14-Jan-1992 12:24:01
Description:

Edited 14-Jan-1992 12:24:01, No Edit Description Entered.

Author : rtse
Date : 16-Oct-1991 14:05:33
Description:

Edited 16-Oct-1991 14:05:33, No Edit Description Entered.

Author : rtse
Date : 21-Jul-1991 22:17:38
Description:

21-Jul-1991 22:18:17

Module CREATION.

INPUT SIGNALS:

INPUT SIGNAL Type: COMPLEX
Lower Limit: (-1.7E38 -1.7E38)
Upper Limit: (1.7E38 1.7E38)

The complex input signal which is to be convolved with the impulse response of the fading multipath fading channel.

OUTPUT SIGNALS:

SIGNAL AFTER MULTIPATH CHANNEL FADING Type: COMPLEX
Lower Limit: (-1.7E38 -1.7E38)
Upper Limit: (1.7E38 1.7E38)

The resulting signal from the time domain convolution between the input signal and the multipath fading channel impulse response (65 samples long, with a sample every 7.8125 n seconds). Note that the software packaged used to create the channel impulse responses used sampling periods of 7.8125E-9 seconds.

PARAMETERS:

CHANNEL NUMBER Type: INTEGER
Lower Limit: 1
Upper Limit: 50

The channel file to read from to get the impulse response.

CHANNEL START NUMBER Type: INTEGER
Lower Limit: 1
Upper Limit: 19

This value determines which impulse response within a channel impulse response file to start at.

UPDATE PERIOD Type: REAL
Lower Limit: 3.0E-39
Upper Limit: 1.7E38

This is the time period between changes in the impulse response. That is, after each period equal to the entered value, a new impulse response will be read from the file.

The entered value should be an integer multiple of 'dt'.

VECTOR LENGTH Type: INTEGER
Lower Limit: 1
Upper Limit: 65

This value indicates the number of samples in each impulse response. The maximum length is 65.

MODULES USED IN BLOCK DIAGRAM:
- DOT PRODUCT
- SERIAL_TO PARALLEL
- VECTOR_ CONJUGATE
- IMPULSE RESPONSES FROM FILE

PARAMETER VALUES FOR INSTANCES IN BLOCK DIAGRAM:

IMPULSE RESPONSES FROM FILE (key 1)
- USER == $CHANNEL NUMBER
- CHANNEL START NUMBER == $CHANNEL START NUMBER
- UPDATE PERIOD == $UPDATE PERIOD
- VECTOR LENGTH == $VECTOR LENGTH

VECTOR_ CONJUGATE (key 2)
- VECTOR LENGTH == $VECTOR LENGTH

SERIAL_TO PARALLEL (key 3)
- OUTPUT VECTOR LENGTH == $VECTOR LENGTH

DOT PRODUCT (key 4)
- VECTOR LENGTH == $VECTOR LENGTH

INITIALIZATION CODE:
- (none)

Figure A7. Primitive block diagram for channel impulse generator.

MODULE NAME: IMPULSE RESPONSES FROM FILE
GROUP: VECTORS // *COMPOSE/DECOMPOSE*
DATABASE: /home/suns/rtse/model_db/
AUTHOR: rtse
CREATION DATE: 17-Jul-1991 14:29:48

DESCRIPTION:

This module reads the impulse responses from a file 'fc#a' where # corresponds to a one or two digit number which should in turn correspond to the USER parameter. A maximum of 50 users is allowed.

The input file should consist of 4 columns of format f10.6 f12.6 e14.6 and e14.6. Where the columns correspond to distance,time,real response, and imaginary response.

The length of the impulse response can vary between 1 and 65 samples.

The parameter 'CHANNEL START NUMBER' determines which set of impulse responses within a channel impulse response file to begin at.

REVISIONS:

Author : rtse
Date : 17-Jan-1992 11:02:01
Description:

Edited 17-Jan-1992 11:02:01, No Edit Description Entered.

Author : rtse
Date : 9-Dec-1991 10:41:47
Description:

Edited 9-Dec-1991 10:41:47, No Edit Description Entered.

Author : rtse
Date : 16-Oct-1991 14:04:58
Description:

Edited 16-Oct-1991 14:04:58, No Edit Description Entered.

Author : rtse
Date : 17-Jul-1991 14:29:48
Description:

17-Jul-1991 14:30:48
Module CREATION.

INPUT SIGNALS:

(none)

OUTPUT SIGNALS:

OUTPUT Type: COMPLEX
Lower Limit: (-1.7E38 -1.7E38)
Upper Limit: (1.7E38 1.7E38)
Vector Length: (VECTOR LENGTH)

This output is the vector containing the impulse response. This vector may be of length 1 to 65.

PARAMETERS:

USER Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

The user number in a multiple access network.

CHANNEL START NUMBER Type: INTEGER
Lower Limit: 1
Upper Limit: 19

This value determines which impulse response within a channel impulse response file to start at.

UPDATE PERIOD Type: REAL
Lower Limit: 3.0E-39
Upper Limit: 1.7E38

This is the time period between changes in the impulse response. That is, after each period equal to the entered value, a new impulse response will be read from the file.

The entered value should be an integer multiple of 'dt'.

VECTOR LENGTH Type: INTEGER
Lower Limit: 1
Upper Limit: 65

This value indicates the number of samples in each impulse response. The maximum length is 65.

MODULES USED IN BLOCK DIAGRAM:

(none - This module is a BOSS Primitive)

INITIALIZATION CODE:
(none)

```
C            ROUTINE_IMPULSE_RESPONSE_FROM_FILE_PRIM

C     This routine reads one of the 50 files containing numerical data
C     of channel impulse responses.  The form of the input should be
C     'time, distance, real response, imaginary response'.  This routine
C     changes the numerical data into complex vectors of length 'len' and
C     outputs the vectors.

C     'out' is the output vector
C     'user' is a parameter which is used to decide which file to read
C     'h' is a vector to be stored in memory
C     'ch_per' is a parameter which is used to determine when to update
C         the memory elements 'h' by rereading the file
C     'count' is a counter which keeps track of where we are in the file
C     'chnl_start' is a parameter which determines which impulse response
C         of a channel impulse response file to start at
C     'len' is the length of the impulse response (max = 65)

      subroutine imp(out,user,h,ch_per,count,chnl_start,len)
      implicit none
      real dist,time,ch_per,mxm
      complex out(*),h(*)
      integer user,i,lun,no1,count,ii,delay,delaymem,chnl_start,len

      include '/bossdir/system/BOSSFORTRAN.INC'


      if (curtime.lt.dt) then
        count=chnl_start-1
      endif

      if (curtime/ch_per - int(curtime/ch_per).lt.0.9*dt/ch_per) then
        call libget_lun(lun)
        no1 = lun
        go to(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,
     &        21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,
     &        37,38,39,40,41,42,43,44,45,46,47,48,49,50),user
    1   open(unit=no1,file='fc1a',status='old')
        go to 99
    2   open(unit=no1,file='fc2a',status='old')
        go to 99
    3   open(unit=no1,file='fc3a',status='old')
        go to 99
    4   open(unit=no1,file='fc4a',status='old')
        go to 99
    5   open(unit=no1,file='fc5a',status='old')
        go to 99
    6   open(unit=no1,file='fc6a',status='old')
        go to 99
    7   open(unit=no1,file='fc7a',status='old')
        go to 99
    8   open(unit=no1,file='fc8a',status='old')
        go to 99
    9   open(unit=no1,file='fc9a',status='old')
        go to 99
   10   open(unit=no1,file='fc10a',status='old')
        go to 99
   11   open(unit=no1,file='fc11a',status='old')
        go to 99
   12   open(unit=no1,file='fc12a',status='old')
```

```
      go to 99
13    open(unit=no1,file='fc13a',status='old')
      go to 99
14    open(unit=no1,file='fc14a',status='old')
      go to 99
15    open(unit=no1,file='fc15a',status='old')
      go to 99
16    open(unit=no1,file='fc16a',status='old')
      go to 99
17    open(unit=no1,file='fc17a',status='old')
      go to 99
18    open(unit=no1,file='fc18a',status='old')
      go to 99
19    open(unit=no1,file='fc19a',status='old')
      go to 99
20    open(unit=no1,file='fc20a',status='old')
      go to 99
21    open(unit=no1,file='fc21a',status='old')
      go to 99
22    open(unit=no1,file='fc22a',status='old')
      go to 99
23    open(unit=no1,file='fc23a',status='old')
      go to 99
24    open(unit=no1,file='fc24a',status='old')
      go to 99
25    open(unit=no1,file='fc25a',status='old')
      go to 99
26    open(unit=no1,file='fc26a',status='old')
      go to 99
27    open(unit=no1,file='fc27a',status='old')
      go to 99
28    open(unit=no1,file='fc28a',status='old')
      go to 99
29    open(unit=no1,file='fc29a',status='old')
      go to 99
30    open(unit=no1,file='fc30a',status='old')
      go to 99
31    open(unit=no1,file='fc31a',status='old')
      go to 99
32    open(unit=no1,file='fc32a',status='old')
      go to 99
33    open(unit=no1,file='fc33a',status='old')
      go to 99
34    open(unit=no1,file='fc34a',status='old')
      go to 99
35    open(unit=no1,file='fc35a',status='old')
      go to 99
36    open(unit=no1,file='fc36a',status='old')
      go to 99
37    open(unit=no1,file='fc37a',status='old')
      go to 99
38    open(unit=no1,file='fc38a',status='old')
      go to 99
39    open(unit=no1,file='fc39a',status='old')
      go to 99
40    open(unit=no1,file='fc40a',status='old')
      go to 99
41    open(unit=no1,file='fc41a',status='old')
      go to 99
42    open(unit=no1,file='fc42a',status='old')
```

```
      go to 99
 43   open(unit=no1,file='fc43a',status='old')
      go to 99
 44   open(unit=no1,file='fc44a',status='old')
      go to 99
 45   open(unit=no1,file='fc45a',status='old')
      go to 99
 46   open(unit=no1,file='fc46a',status='old')
      go to 99
 47   open(unit=no1,file='fc47a',status='old')
      go to 99
 48   open(unit=no1,file='fc48a',status='old')
      go to 99
 49   open(unit=no1,file='fc49a',status='old')
      go to 99
 50   open(unit=no1,file='fc50a',status='old')
 99   continue

      count = count+1

      do i = 1,count,1
        mxm = 0.0
        do ii = len,1,-1
          read(no1,100) dist,time,h(ii)
100       format(f10.6,f12.6,e14.6,e14.6)
        enddo
      enddo

      close(no1,status='keep')
      call libfree_lun(no1)
    endif

    do i=1,len,1
      out(i) = h(i)
    enddo


    return
    end
```

A1.v RAKE DPSK Demodulator

Figure A8 shows the RAKE DPSK Demodulator. The correlator section takes advantage of third-order multipath diversity by using three delayed versions of the Gold sequence, each synchronized to consecutive versions of the received signal resulting from multipath propagation, to correlate with different portions of the received signal. Each will despread one portion of the signal which has propagated through a multipath channel. The Gold sequence is input from an external source and should be already synchronized to the received signal. The equivalent of the despread interference is input to the module from an external source and is added to the despread signals before they each pass through an integrate and dump filter. The outputs of each filter are sampled and DPSK demodulated before being summed. For DPSK demodulation, the change in phase between the current bit and the last bit must be found to determine if a positive or a negative bit was transmitted. This is accomplished by multiplying the current sample by the complex conjugate of the previous sample. The final summed result is the demodulated signal. The BOSS module allows the diversity order to be varied between 1 and 3 by a user controlled parameter.

The DPSK demodulation is performed by the module DPSK DEMOD (BB,NMF) shown in Figure A9.

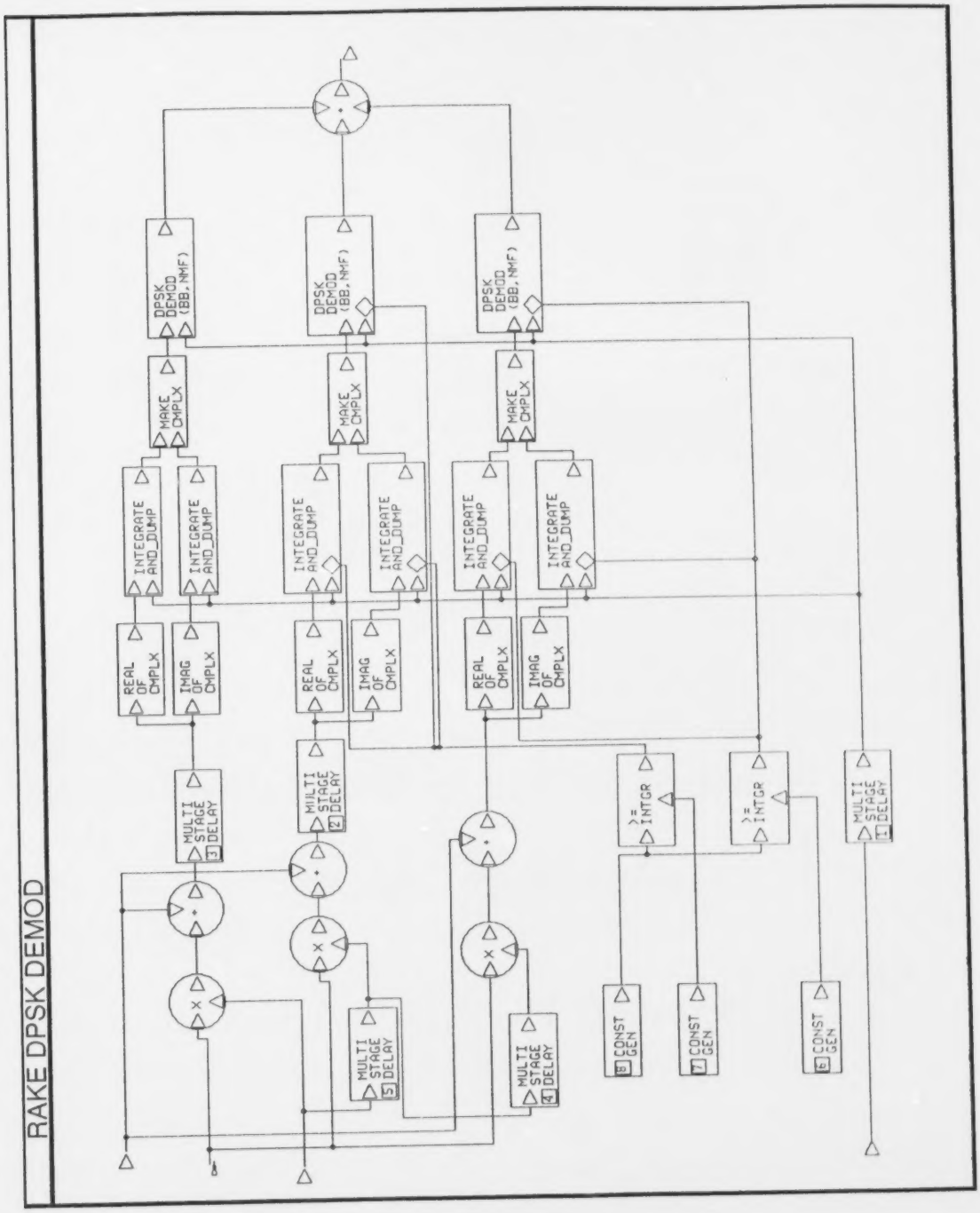


Figure A8. A RAKE receiver for DPSK signals.

MODULE NAME: RAKE DPSK DEMOD
GROUP: DEMODULATORS
DATABASE: /home/suns/rtse/model_db/
AUTHOR: rtse
CREATION DATE: 13-Mar-1992 10:13:33

DESCRIPTION:

This module is a RAKE-based diversity receiver. The diversity inherent in spread spectrum systems is used and each delayed element is fed into a DPSK demodulator. The maximum diversity order for this particular receiver is 3.

A pn code for despreading or descrambling of the signal is available. If one is not used, then a constant of (1.0,0.0) should be input in its place.

The output should be sampled midway into the recovered data signal to ensure that transient responses in the decoders have passed. The output need only be compared to see if it is greater than or less than zero to determine the final logical output (>0 --> true, <0 -->false).

REVISIONS:

Author : rtse
Date : 13-Mar-1992 10:13:33
Description:

13-Mar-1992 10:14:13
Module CREATION.

INPUT SIGNALS:

NOISE AND INTERFERENCE Type: COMPLEX
Lower Limit: (-1.7E38 -1.7E38)
Upper Limit: (1.7E38 1.7E38)

The noise and interference is to be added here. The power

of this input should be equal to the power of the noise and interference after despreading of the spread spectrum signal.

SAMPLING POINT Type: LOGICAL
Lower Limit: NIL
Upper Limit: NIL

This is a logical signal that tells the module when to sample for demodulation.

PN SEQUENCE Type: COMPLEX
Lower Limit: (-1.0 -1.0)
Upper Limit: (1.0 1.0)

The pn sequence used for depsreading the spread spectrum signal.

INPUT SIGNAL Type: COMPLEX
Lower Limit: (-1.7E38 -1.7E38)
Upper Limit: (1.7E38 1.7E38)

This is the received signal after the matched filter.

OUTPUT SIGNALS:

DEMODULATED OUTPUT Type: REAL
Lower Limit: -1.7E38
Upper Limit: 1.7E38

The signal after RAKE DPSK demodulation.

PARAMETERS:

CHIP RATE Type: REAL
Lower Limit: 3.0E-39
Upper Limit: 1.7E38

The chip rate in bits/sec.

DIVERSITY ORDER Type: INTEGER
Lower Limit: 1
Upper Limit: 3

This integer determines the number of demodulators/decoders to use to recover the transmitted data. The minimum diversity order is 1 and the maximum is 3.

MODULES USED IN BLOCK DIAGRAM:

```
>= INTGR
CONST GEN
MULTIPLIER
  INPUT ADDER
IMAG OF CMPLX
REAL OF CMPLX
INTEGRATE AND_DUMP
MAKE CMPLX
DPSK DEMOD (BB,NMF)
MULTI STAGE DELAY
ADDER
```

PARAMETER VALUES FOR INSTANCES IN BLOCK DIAGRAM:

```
MULTI STAGE DELAY (key 1)
   DELAY NUM ==  Round (2.0 / ('CHIP RATE' * 'DT'))

MULTI STAGE DELAY (key 2)
   DELAY NUM ==  Round (1.0 / ('CHIP RATE' * 'DT'))

MULTI STAGE DELAY (key 3)
   DELAY NUM ==  Round (2.0 / ('CHIP RATE' * 'DT'))

MULTI STAGE DELAY (key 4)
   DELAY NUM ==  Round (1.0 / ('CHIP RATE' * 'DT'))

MULTI STAGE DELAY (key 5)
   DELAY NUM ==  Round (1.0 / ('CHIP RATE' * 'DT'))

CONST GEN (key 6)
   CONSTANT VALUE == 3

CONST GEN (key 7)
   CONSTANT VALUE == 2

CONST GEN (key 8)
   CONSTANT VALUE == $DIVERSITY ORDER
```

INITIALIZATION CODE:

(none)

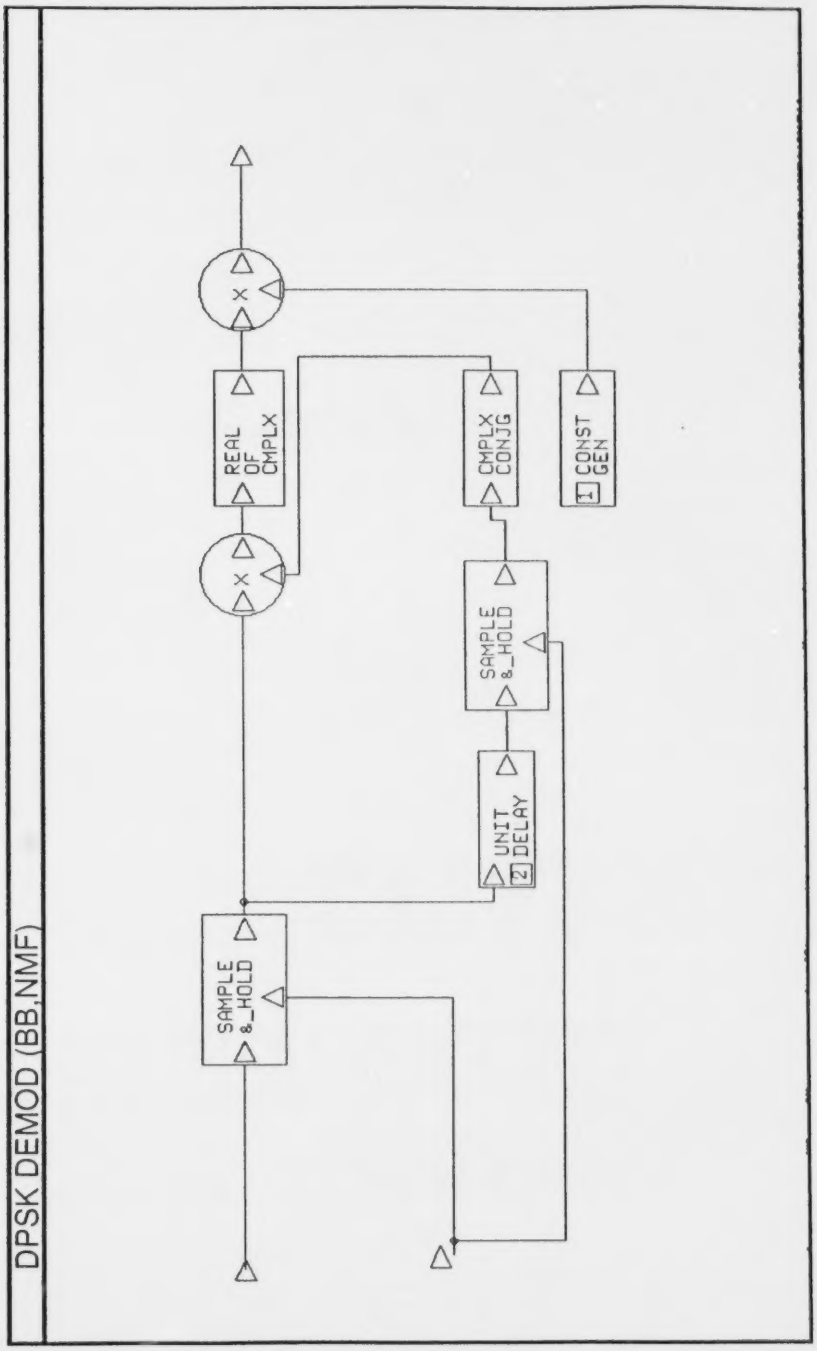


Figure A9. A demodulator for DPSK signals at baseband.

MODULE NAME: DPSK DEMOD (BB,NMF)
GROUP: DEMODULATORS
DATABASE: /home/suns/rtse/model_db/
AUTHOR: rtse
CREATION DATE: 7-Oct-1991 11:22:39

DESCRIPTION:

This modulate recovers a real value which corresponds to the DPSK modulated digital signal which has already been passed through a matched filter.
The module output will be positive if a positive bit was received, and will be negative if a negative bit was received. It is of type real.
A logical impulse signal is required for bit synchronization.

REVISIONS:

Author : rtse
Date : 9-Jan-1992 11:23:31
Description:

Edited 9-Jan-1992 11:23:31, No Edit Description Entered.

Author : rtse
Date : 10-Dec-1991 13:27:57
Description:

Edited 10-Dec-1991 13:27:57, No Edit Description Entered.

Author : rtse
Date : 7-Oct-1991 11:22:39
Description:

7-Oct-1991 11:22:53
Module CREATION.

INPUT SIGNALS:

BIT SYNCHRONIZATION SIGNAL Type: LOGICAL
Lower Limit: NIL
Upper Limit: NIL

This signal should consist of an impulse signal at the beginning of each received data bit. This allows the demodulator to determine when to sample its outputs.

RECEIVED SIGNAL INPUT Type: COMPLEX
Lower Limit: (-1.7E38 -1.7E38)

Upper Limit: (1.7E38 1.7E38)

This should be the baseband DPSK modulated signal which has already passed through a matched filter.

OUTPUT SIGNALS:

DEMODULATED OUTPUT Type: REAL
Lower Limit: -1.7E38
Upper Limit: 1.7E38

This signal represents the demodulated output. If this real number is greater than zero, then the input signal was a positive bit. If it is less than zero, then the input signal was a negative bit.

PARAMETERS:
(none)

MODULES USED IN BLOCK DIAGRAM:
CMPLX CONJG
REAL OF CMPLX
SAMPLE & HOLD
UNIT DELAY
CONST GEN
MULTIPLIER

PARAMETER VALUES FOR INSTANCES IN BLOCK DIAGRAM:

CONST GEN (key 1)
CONSTANT VALUE == -1.0

UNIT DELAY (key 2)
INITIAL VALUE == (0.0 , 0.0)

INITIALIZATION CODE:
(none)

A1.vi Viterbi Decoder (or Orthogonal Convolutional Decoder)

Figure A10 shows the block diagram of the orthogonal convolutional decoder (OC_DECODER). One input is for the hard-decision outputs of the RAKE DPSK demodulator and the other is for the timing signal which tells the decoder when to sample those outputs. The code ROUTINE_VITERBI_DECODER_PRIM performs maximum-likelihood decoding using hard-decision inputs based on Euclidean distances for orthogonal convolutional codes. Parameters allow the user to set the code constraint length K, the maximum path memory, and the incoming bit rate.

The program reads in all 2^K trellis states of the convolutional code and stores them in memory when initialized. Once enabled, the decoder reads the input bits. Every 2^K bits, it will determine branch metrics and path metrics and then flag the surviving path for each state. The best path among these is chosen as the correct one (at this point) and the trellis state which is connected to the oldest branch is chosen as the decoded state. Because orthogonal convolutional codes are used, the first bit of this final state determines the decoded input. If the first bit is a +1, then a +1 is the decoded output. If it is a -1, then a -1 is the decoded output.

The decoder features a truncated path memory. A parameter is available to the user to specify this memory size. This parameter should equal 5 times the constraint length of the code (see Subsection 2.2.2. for details).

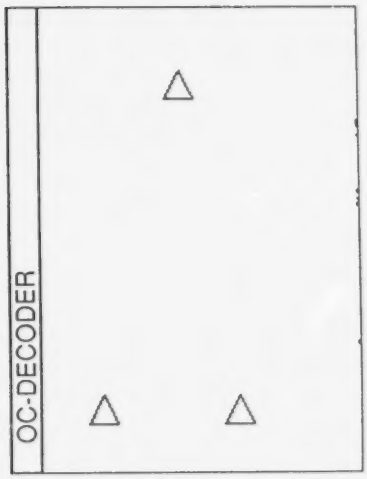


Figure A10. Primitive block diagram for the orthogonal convolutional decoder.

```
MODULE NAME: OC-DECODER
GROUP: SOURCE ENCODERS/DECODERS
DATABASE: /home/suns/rtse/model_db/
AUTHOR: rtse
CREATION DATE: 12-Aug-1991 13:18:48

DESCRIPTION:

   The orthogonal convolutional decoder will recover the
   original data bits from the coded signal using
   maximum likelihood sequence estimation.  The input
   and output are INTEGER values of (+1 or -1).

   ****************************************************
   If taking the output straight out of the OC ENCODER,
   the polarity of the bits should be reversed first!!
   ****************************************************

   A logical signal SAMPLE SIGNAL is used to tell the
   module when to sample the input bits.  The first
    .true. SAMPLE SIGNAL should correspond to the first
   code symbol of the all zero encoding state.

   Note that for the input bits, a +1 actually corresponds
   to a coded 0 bit and a -1 corresponds to a coded 1 bit!!

REVISIONS:


   Author      : rtse
   Date        : 17-Mar-1992 11:33:11
   Description:

   Edited 17-Mar-1992 11:33:11, No Edit Description Entered.

   Author      : rtse
   Date        : 12-Feb-1992 10:53:21
   Description:

   Edited 12-Feb-1992 10:53:21, No Edit Description Entered.

   Author      : rtse
   Date        : 21-Aug-1991 19:45:27
   Description:

   Edited 21-Aug-1991 19:45:27, No Edit Description Entered.

   Author      : rtse
   Date        : 12-Aug-1991 13:18:48
   Description:


   12-Aug-1991 13:19:01
     Module CREATION.
```

INPUT SIGNALS:

SAMPLE SIGNAL Type: LOGICAL
Lower Limit: NIL
Upper Limit: NIL

This signal should be true only when the input bit is to be sampled.

INPUT Type: INTEGER
Lower Limit: -1
Upper Limit: 1

This is the orthogonal convolutionally encoded data that is to be decoded to recover the original data. The input should be in the form of binary integers of values (+1/-1).

OUTPUT SIGNALS:

OUTPUT Type: INTEGER
Lower Limit: -1
Upper Limit: 1

The output is in the form of binary integers (+1,-1). The output corresponds to the maximum likelihood sequence estimation of the original input data.

PARAMETERS:

PATH MEMORY Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

The truncation length of the decoder. This value should equal 5 times the code constraint length for minimal loss of decoding gain.

K Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

The constraint length of the orthogonal convolutional encoder.

VALUE OF 2^(2*K) Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

The value of 2^(2*K). This value is used to allocate memory to store all the valid orthogonal convolutional code words.

VALUE OF 2^K Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

The value of 2^K. This value is used to allocate memory for the array which stores the input symbols.

MODULES USED IN BLOCK DIAGRAM:

(none - This module is a BOSS Primitive)

INITIALIZATION CODE:
(none)

```
C                ROUTINE_VITERBI_DECODER_PRIM

C       This program takes the orthogonal convolutional coded input and
C      restores the original uncoded data using the typical ACS operaton
C      of conventional Viterbi decoders.  The output is weighted by
C      multiplying the binary output value by its best path metric.
C       The path memory length is limited to the user specified parameter
C      value.  This value should be approximately 5 times the constraint
C      length to assure a high probability of path mergence at the
C      truncation point.
C       The available codes are read from the file 'all_codes.dat', and
C      these are used in the ACS operations.  The file 'all_codes.dat' is
C      generated by the 'C' code 'gen_all.c'.
C       This decoder assumes that the input bits are coded in polar
C      form (+1/-1) where -1 corresponds to a data '1' and +1 corresponds
C      to a data '0'.  The encoded symbols must be inverted (+1 -> -1 and
C      -1 -> +1) before entering the decoder for it to work properly.
C      IT IS IMPORTANT TO REMEMBER THE INVERSION OF THE POLARITY OF THESE
C      INPUT BITS!!!!  Also, the starting state is assumed to be the all
C      all zero state;  that is, the encoding shift register contains all
C      all zeros.

C      'in' is the input signal to be decoded (input signal)
C      'sample' is the signal which indicates that another symbol should
C         be read (input signal)
C      'pathmem' is the length of the path memory (parameter)
C      'K' is the code constraint length (parameter)
C      'twotwok' should equal 2^(2*K):used to allocate array code() (parameter)
C      'twok' should equal 2^K:used to allocate array chip() (parameter)
C      'num' keeps track of which coded  symbol (chip) the decoder
C          is processing (memory)
C      'chip' is one coded symbol (memory array of size 2^K)
C      'start' is a flag which indicates if the decoder is just
C         starting its decoding (memory)
C      'pthend' is the memory variable which keeps track of where
C         in the path the decoder is at (memory)
C      'thrshflg' is a flag used to see if the decoder has read
C         enough bits to satisfy the path memory (memory)
C      'total' is equal to 2^K (memory)
C      'metmem' is the memory array which saves each branch metric (memory)
C      'pthsum' is the memory array which stores the surviving path
C         metric for the current stage and the previous stage
C      'metpth' is the memory array which stores the state from which
C         the current state branched from
C      'out' is the weighted decoded output

       subroutine vitdec(in,sample,pathmem,K,twotwok,twok,num,code,
      &                  chip,start,pthend,thrshflg,total,metmem,
      &                  pthsum,metpth,out)

       implicit none
       integer i,j,prev0,prev1,prev,fstate,start,point,lun,l
       integer K,total,pathmem,num,met,twok,met0t,met1t
       integer pthend,lastend,num,thrshflg,bit,max
       integer chip(*),pthsum(1025,2),twotwok
       integer code(*),metpth(1025,51),metmem(1025,51)
       integer in,out
       logical*1 sample

       include '/bossdir/system/BOSSFORTRAN.INC'
```

```
C     Initialization:  all valid codewords are read and counter and flags
C                      are reset.
      if (curtime.lt.dt) then
        call libget_lun(lun)
        go to (1,2,3,4,5,6,7,8,9,10),k
    1   open(unit=lun,file='non_exist.dat',status='old')
        goto 11
    2   open(unit=lun,file='non_exist.dat',status='old')
        goto 11
    3   open(unit=lun,file='all_codes8.dat',status='old')
        goto 11
    4   open(unit=lun,file='all_codes16.dat',status='old')
        goto 11
    5   open(unit=lun,file='all_codes32.dat',status='old')
        goto 11
    6   open(unit=lun,file='all_codes64.dat',status='old')
        goto 11
    7   open(unit=lun,file='all_codes128.dat',status='old')
        goto 11
    8   open(unit=lun,file='all_codes256.dat',status='old')
        goto 11
    9   open(unit=lun,file='all_codes512.dat',status='old')
        goto 11
   10   open(unit=lun,file='all_codes1024.dat',status='old')
   11   continue
        read(lun,*) l
        read(lun,*) total
        do i = 1,total,1
          read(lun,*) num
          do j = 1,total,1
            l = (i-1)*total+j
            read(lun,*) code(l)
          enddo
        enddo
        close(lun,status='keep')
        call libfree_lun(lun)
        start = 0
        pthend = 1
        thrshflg = 0
        num = 1
      endif


C     Read input data whenever the external 'sample' signal is high.
C     If the number of code symbols has not yet formed a complete
C     codeword, then return from this subroutine.
      if (sample) then
        chip(num) = in
        if (num.ne.total) then
          num = num+1
          goto 9999
        else
          num = 1
        endif
      else
        goto 9999
      endif
```

```
C     Set the counter which keeps track of which branch stage the
C     decoder is at.
      if (pthend.eq.pathmem) then
        pthend = 1
        lastend =  pathmem
      else
        lastend = pthend
        pthend = pthend+1
      endif


C     Check to see if the number of input bits read at the start
C     are equal to the pathmem delay.  If not, then set the flag
C     so that the decoder does not trace the paths back to decode
C     yet as more bits must be read before the ACS decoding operations
C     can proceed.
      if (thrshflg.eq.0) then
         if (pthend.eq.1) then
             thrshflg = 1
         endif
      endif

C     Remove the value of the earliest metric from the path metric sums.
      if (thrshflg.eq.1) then
      do i = 1,total,1
         prev = i
         do j = 1,pathmem-1,1
           point = lastend-j
           if (point.lt.1) then
             point = point+pathmem
           endif
           if (point.ne.pthend) then
             prev = metpth(prev,point)
           endif
         enddo
         pthsum(i,1) = pthsum(i,2)-metmem(prev,point)
      enddo
      endif

C     Calculate branch and path metric sums.
      do i = 1,total,1
        if (mod(i,2).eq.0) then
          prev1 = (i+total)/2
          prev0 = i/2
        else
          prev1 = (i+1+total)/2
          prev0 = (i+1)/2
        endif
        met = 0
        do j = 1,total,1
          met = met + code((i-1)*total+j)*chip(j)
        enddo
        met0t = met + pthsum(prev0,1)
        met1t = met + pthsum(prev1,1)
        if (thrshflg.ne.1.and.pthend.eq.2) then
          met1t = 0
        endif
        if (met0t.gt.met1t) then
          metpth(i,pthend) = prev0
          pthsum(i,2) = met0t
```

```
            metmem(i,pthend) = met
          else
            metpth(i,pthend) = prev1
            pthsum(i,2) = met1t
            metmem(i,pthend) = met
          endif
        enddo
        do i = 1,total,1
         pthsum(i,1) = pthsum(i,2)
        enddo

C       Find the path that has the largest metric and then trace this
C       path back to find the decoded data bit.
        if (thrshflg.eq.1) then
        max = 0
        do i = 1,total,1
          if (pthsum(i,2).gt.max) then
            max = pthsum(i,2)
            bit = i
          endif
        enddo

        prev = bit
        do i = 1,pathmem,1
          point = pthend-i+1
          if (point.lt.1) then
            point =  point + pathmem
          endif
          prev = metpth(prev,point)
        enddo
        fstate = prev

        if (mod(fstate,2).eq.0) then
           out = 1*max
        else
           out = -1*max
        endif

        endif
 9999   return
        end
```

A1.vii Error Counter

The Error Counter is shown in Figure A11. It compares two different input binary signals and records the number of differences between them. Time delays between the two signals can be accounted for as there are two other input signals (one for the reference signal and one for the test signal) which tell the module when to begin sampling. Parameters are available to allow the user to specify how often (in number of bits compared) he would like the results to be written out. The module BER_DETECTOR WITH TERMINATE SIGNAL shown in Figure A12 is a BOSS primitive whose code is given as ROUTINE_BER_WITH_TERM_PRIM. This module compares the two input binary signals and counts the total number of bits and the total number of errors observed. The results are written to the file 'ber*xx*.file' where *xx* is a number from 1 to 20 inclusive which is specified by the user. Results are written whenever an error is encountered. Also, the user can have the results written whenever a certain number of bits have been counted. The results given by the module are: the simulation time that the result was written, the number of bits compared, the number of errors detected, and the bit error rate.

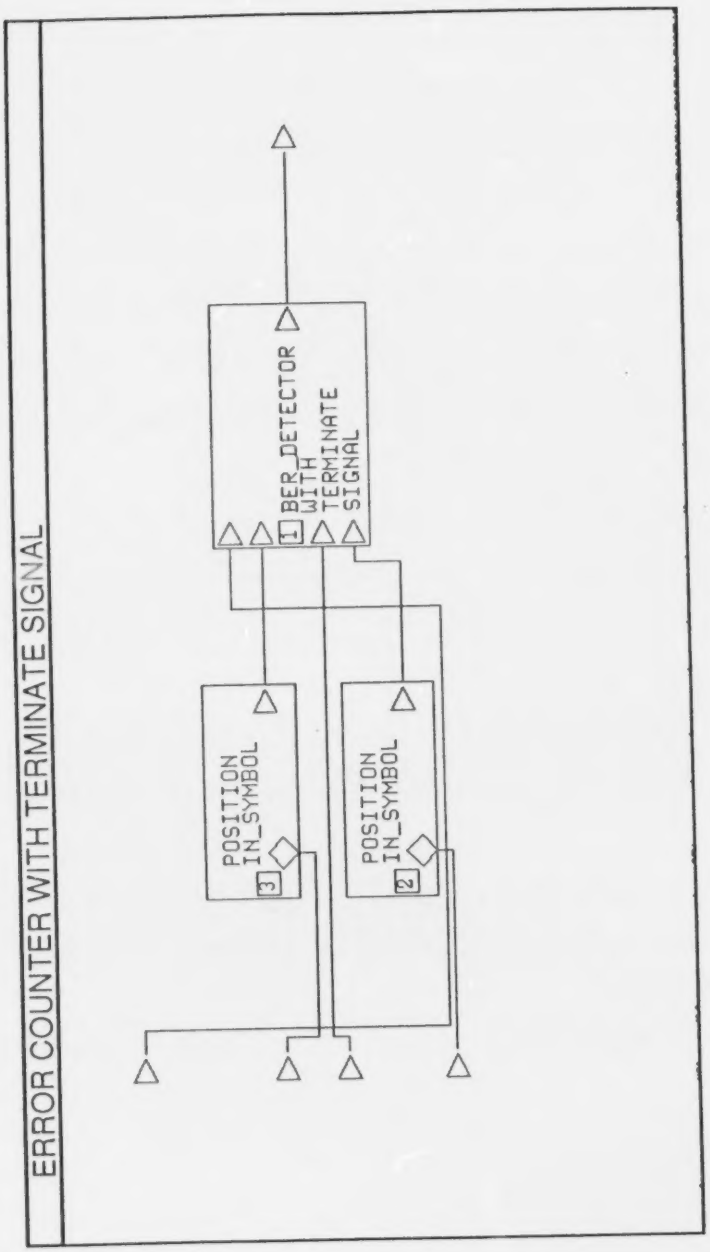


Figure A11. The error counter.

MODULE NAME: ERROR COUNTER WITH TERMINATE SIGNAL
GROUP: MISCELLANEOUS
DATABASE: /home/suns/rtse/model_db/
AUTHOR: rtse
CREATION DATE: 13-Apr-1992 17:59:35

DESCRIPTION:

This module counts the number of differences between two streams of data; a reference stream and a test stream. Running results are written whenever the number of bits tested is equal to the value in the parameter #BITS FOR A FILE UPDATE. The simulation is terminated when the number of errors detected equals the parameter #ERRORS TO TERMINATE SIMULATION. The time, total number of bits tested, total number of errors, and the bit error rate are given.

The results are written to the file 'ber##.file' where ## represents a number between 1 and 20 inclusive which is entered as a parameter.

REVISIONS:

Author : rtse
Date : 13-Apr-1992 20:30:39
Description:

Edited 13-Apr-1992 20:30:39, No Edit Description Entered.

Author : rtse
Date : 13-Apr-1992 20:22:07
Description:

Edited 13-Apr-1992 20:22:07, No Edit Description Entered.

Author : rtse
Date : 13-Apr-1992 17:59:35
Description:

13-Apr-1992 18:00:06
Module CREATION.

INPUT SIGNALS:

TEST START SIGNAL Type: LOGICAL
Lower Limit: NIL
Upper Limit: NIL

This signal should go from .false. to .true. when the test data starts.

REFERENCE START SIGNAL Type: LOGICAL
Lower Limit: NIL
Upper Limit: NIL

This signal should go from .false. to .true. when the reference data begins.

TEST DATA Type: LOGICAL
Lower Limit: NIL
Upper Limit: NIL

The data to be tested for bit errors.

REFERENCE DATA Type: LOGICAL
Lower Limit: NIL
Upper Limit: NIL

The data to be used as the reference when testing for bit errors.

OUTPUT SIGNALS:

TERMINATE SIMULATION SIGNAL Type: LOGICAL
Lower Limit: NIL
Upper Limit: NIL

This signal will turn .true. when the simulation is to be terminated.

PARAMETERS:

FILE NUMBER Type: INTEGER
Lower Limit: 1
Upper Limit: 20

The value entered here determines which file the output will be written to. Valid entries are integers from 1 to 20 inclusive. The output file is 'ber##.file' where ## represents the parameter value.

#ERRORS TO TERMINATE SIMULATION Type: INTEGER
Lower Limit: 0
Upper Limit: 2147483647

When this number of errors is detected, the terminate simulation signal output will go .true..

#BITS FOR A FILE UPDATE Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

The number of bits that are compared before the file inclusive).

DATA RATE Type: REAL
Lower Limit: 2.999998E-39
Upper Limit: 1.7E38

The rate of the binary data to be compared (in bits/sec).

MAX DELAY (#BITS) Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

This parameter is used to allocate storage memory for the reference signal as there is usually a time delay between the reference and test signals (due to mod/demod, processing, etc.). The value entered should be an integer EQUAL TO or GREATER than the number of bit periods that the two binary signals are separated by.

MODULES USED IN BLOCK DIAGRAM:
POSITION IN_SYMBOL
BER_DETECTOR WITH TERMINATE SIGNAL

PARAMETER VALUES FOR INSTANCES IN BLOCK DIAGRAM:

BER_DETECTOR WITH TERMINATE SIGNAL (key 1)
#ERRORS TO TERMINATE SIMULATION == $#ERRORS TO TERMINATE SIMULATION
FILE NUMBER == $FILE NUMBER
#BITS FOR A FILE UPDATE == $#BITS FOR A FILE UPDATE
MAX DELAY (#BITS) == $MAX DELAY (#BITS)

POSITION IN_SYMBOL (key 2)
TIME DELAY TO INPUT (SEC) == 0
SYMBOL FRAC FOR SAMPLE TIME == 0
SYMBOL RATE (HZ) == $DATA RATE

POSITION IN_SYMBOL (key 3)
TIME DELAY TO INPUT (SEC) == 0
SYMBOL FRAC FOR SAMPLE TIME == 0
SYMBOL RATE (HZ) == $DATA RATE

INITIALIZATION CODE:
(none)

Figure A12. Primitive block diagram for the bit error detector.

MODULE NAME: BER_DETECTOR WITH TERMINATE SIGNAL
GROUP: MISCELLANEOUS
DATABASE: /home/suns/rtse/model_db/
AUTHOR: rtse
CREATION DATE: 13-Apr-1992 17:34:31

DESCRIPTION:

This module takes two input binary signals, one reference and one test, and determines the number of different bits there are between them, and thus, obtain the bit error rate. The BER is calculated and written to a file whenever a certain number of bits are compared (a user parameter). A 'terminate simulation' signal will go .true. when a certain number of errors are detected (another user paramter). This output is stored in the file 'ber#.file' where # is an integer between 1 and 20 (inclusive).

In most systems, there is a time delay between the reference and the test signals. Thus, two logical signals are needed to signal the start of each data stream. When REFERENCE SAMPLE SIGNAL and TEST SAMPLE SIGNAL are high, the module will read the reference data and test data streams respectively.

REVISIONS:

Author : rtse
Date : 13-Apr-1992 20:29:52
Description:

Edited 13-Apr-1992 20:29:52, No Edit Description Entered.

Author : rtse
Date : 13-Apr-1992 20:18:32
Description:

Edited 13-Apr-1992 20:18:32, No Edit Description Entered.

Author : rtse
Date : 13-Apr-1992 17:34:31
Description:

13-Apr-1992 17:34:35
Module CREATION.

INPUT SIGNALS:

TEST SAMPLE SIGNAL Type: LOGICAL
Lower Limit: NIL
Upper Limit: NIL

This signal should be true only when a sample of the test signal is to be taken.

REFERENCE SAMPLE SIGNAL Type: LOGICAL
Lower Limit: NIL
Upper Limit: NIL

This signal should be true only when a sample of the reference bit stream is to be taken.

TEST DATA Type: LOGICAL
Lower Limit: NIL
Upper Limit: NIL

The recovered binary data to be compared to the original so that the number of errors and the error rate can be determined.

REFERENCE DATA Type: LOGICAL
Lower Limit: NIL
Upper Limit: NIL

The original information bits that would ideally be recovered at the receiver.

OUTPUT SIGNALS:

TERMINATE SIMULATION SIGNAL Type: LOGICAL
Lower Limit: NIL
Upper Limit: NIL

This signal will go .true. to terminate the simulation when the number of errors detected equals the number entered to the parameter '#ERROR TO TERMINATE SIMULATION'.

PARAMETERS:

#ERRORS TO TERMINATE SIMULATION Type: INTEGER
Lower Limit: 0
Upper Limit: 2147483647

When this number of errors is detected, the terminate simulation signal output will go .true..

FILE NUMBER Type: INTEGER
Lower Limit: 1
Upper Limit: 20

The value entered here determines which file the output will be written to. Valid entries are integers from 1 to 20 inclusive. The output file is 'ber##.file' where ## represents the parameter value.

#BITS FOR A FILE UPDATE Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

The number of bits that are compared before the file inclusive).

MAX DELAY (#BITS) Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

This parameter is used to allocate storage memory for the reference signal as there is usually a time delay between the reference and test signals (due to mod/demod, processing, etc.). The value entered should be an integer EQUAL TO or GREATER than the number of bit periods that the two binary signals are separated by.

MODULES USED IN BLOCK DIAGRAM:

(none - This module is a BOSS Primitive)

INITIALIZATION CODE:
(none)

```
C                ROUTINE_BER_WITH_TERM_PRIM

C     This module reads in two separate streams of logical data.  Each data
C     stream has its own sampling signal which tells it when to sample.
C     The module accumulates the number of total bits and the number of
C     different bits (samples) between the two streams.  From this, the bit
C     error rate over the time period can be found.
C     It is expected that in any system, there will be a time delay between the
C     original reference bits and the received and decoded bits.  The maximum
C     delay (in number of bits) must be specified in the parameter 'maxdelay'.
C     This module writes its most up to date results every time the 'wrt'signal
C     is high.  The results are written to a file 'ber##.file' where '##' is
C     a number in the range 1 through 20 inclusive.

      subroutine biterr(refsig,testsig,rsampsig,tsampsig,delay,
     &                  refno,testno,error,filenum,maxdelay,refm,
     &                  numerrterm,numbitwrt,termsim,totbits)

      implicit none
      integer delay,refno,testno,numerrterm,numbitwrt
      integer error,totbits,filenum,maxdelay,no
      logical*1 rsampsig,tsampsig,testsig,refsig,refm(*),wrt,termsim
      logical*1 tstart

      include '/bossdir/system/BOSSFORTRAN.INC'

      if (curtime.lt.dt) then
         delay = maxdelay
         refno = 0
         testno = 0
         error = 0
         totbits = 0
         termsim = .false.
         wrt = .true.
         call fileinit(filenum,no)
         close(no)
         call lib$free_lun(no)
      endif

      if (rsampsig.eq..true.) then
          refno = refno+1
          if (refno.gt.delay) then
            refno=1
          endif
          refm(refno) = refsig
      endif

      if (tsampsig.eq..true.) then
         if (tstart.eq.0) then
           delay = refno
           tstart = 1
         endif
         testno = testno + 1
         if (testno.gt.delay) then
           testno = 1
         endif
         if (refm(testno).ne.testsig) then
           error = error + 1
           wrt = .true.
```

```
      endif
      totbits = totbits + 1
      if (mod(float(totbits),float(numbitwrt)).lt.0.1) then
         wrt = .true.
      endif
      if (error.eq.numerrterm) then
        termsim = .true.
        wrt = .true.
      endif
    endif


    if (wrt.eq..true.) then
      call file(filenum,no)
      write(no,200) curtime
      write(no,210) totbits
      write(no,220) error
      write(no,230) float(error)/float(totbits+1.0E-9)
      if (abs(curtime-stop_time).lt.dt) then
        write(no,*) 'END OF SIMULATION'
      endif
      close(no)
      call lib$free_lun(no)
      wrt = .false.
    endif
200 format('Curtime: ',e15.8)
210 format('Total number of bits tested: ',i7)
220 format('Total number of errors detected: ',i7)
230 format('Bit error rate: ',f9.7)

    return
    end

    subroutine file(filenum,no)
      integer filenum,no,lun
      call lib$get_lun(lun)
      no = lun
      go to (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,
     &       18,19,20),filenum
  1   open(unit=lun,file='ber1.file',access='append')
      goto 21
  2   open(unit=lun,file='ber2.file',access='append')
      goto 21
  3   open(unit=lun,file='ber3.file',access='append')
      goto 21
  4   open(unit=lun,file='ber4.file',access='append')
      goto 21
  5   open(unit=lun,file='ber5.file',access='append')
      goto 21
  6   open(unit=lun,file='ber6.file',access='append')
      goto 21
  7   open(unit=lun,file='ber7.file',access='append')
      goto 21
  8   open(unit=lun,file='ber8.file',access='append')
      goto 21
  9   open(unit=lun,file='ber9.file',access='append')
      goto 21
 10   open(unit=lun,file='ber10.file',access='append')
      goto 21
 11   open(unit=lun,file='ber11.file',access='append')
```

```
      goto 21
12    open(unit=lun,file='ber12.file',access='append')
      goto 21
13    open(unit=lun,file='ber13.file',access='append')
      goto 21
14    open(unit=lun,file='ber14.file',access='append')
      goto 21
15    open(unit=lun,file='ber15.file',access='append')
      goto 21
16    open(unit=lun,file='ber16.file',access='append')
      goto 21
17    open(unit=lun,file='ber17.file',access='append')
      goto 21
18    open(unit=lun,file='ber18.file',access='append')
      goto 21
19    open(unit=lun,file='ber19.file',access='append')
      goto 21
20    open(unit=lun,file='ber20.file',access='append')
21    continue
      return
      end

   subroutine fileinit(filenum,no)
      integer filenum,no,lun
      call lib$get_lun(lun)
      no = lun
      go to (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,
     &       19,20),filenum
 1    open(unit=lun,file='ber1.file')
      goto 21
 2    open(unit=lun,file='ber2.file')
      goto 21
 3    open(unit=lun,file='ber3.file')
      goto 21
 4    open(unit=lun,file='ber4.file')
      goto 21
 5    open(unit=lun,file='ber5.file')
      goto 21
 6    open(unit=lun,file='ber6.file')
      goto 21
 7    open(unit=lun,file='ber7.file')
      goto 21
 8    open(unit=lun,file='ber8.file')
      goto 21
 9    open(unit=lun,file='ber9.file')
      goto 21
10    open(unit=lun,file='ber10.file')
      goto 21
11    open(unit=lun,file='ber11.file')
      goto 21
12    open(unit=lun,file='ber12.file')
      goto 21
13    open(unit=lun,file='ber13.file')
      goto 21
14    open(unit=lun,file='ber14.file')
      goto 21
15    open(unit=lun,file='ber15.file')
      goto 21
16    open(unit=lun,file='ber16.file')
      goto 21
```

```
17    open(unit=lun,file='ber17.file')
      goto 21
18    open(unit=lun,file='ber18.file')
      goto 21
19    open(unit=lun,file='ber19.file')
      goto 21
20    open(unit=lun,file='ber20.file')
21    continue
      return
      end
```

A1.viii SSMA Indoor Wireless System

The SSMA indoor wireless system shown in Figure A13 is made up of all the aforementioned BOSS modules, some standard BOSS modules, and some linking modules. It is used to simulate the coded SSMA system. The system used to simulate the uncoded SSMA system is shown in Figure A14. The systems are described in Chapter 4. The same initialization code is used for both the coded and uncoded systems and is given as ROUTINE_SYSTEM_INIT at the end of this section. This code finds the optimal sampling point for the receiver. This optimal sampling point is the point where the maximum eye opening can be found at the demodulator. The code also determines what the signal to interference level is from the number of interferers and the code rate used.

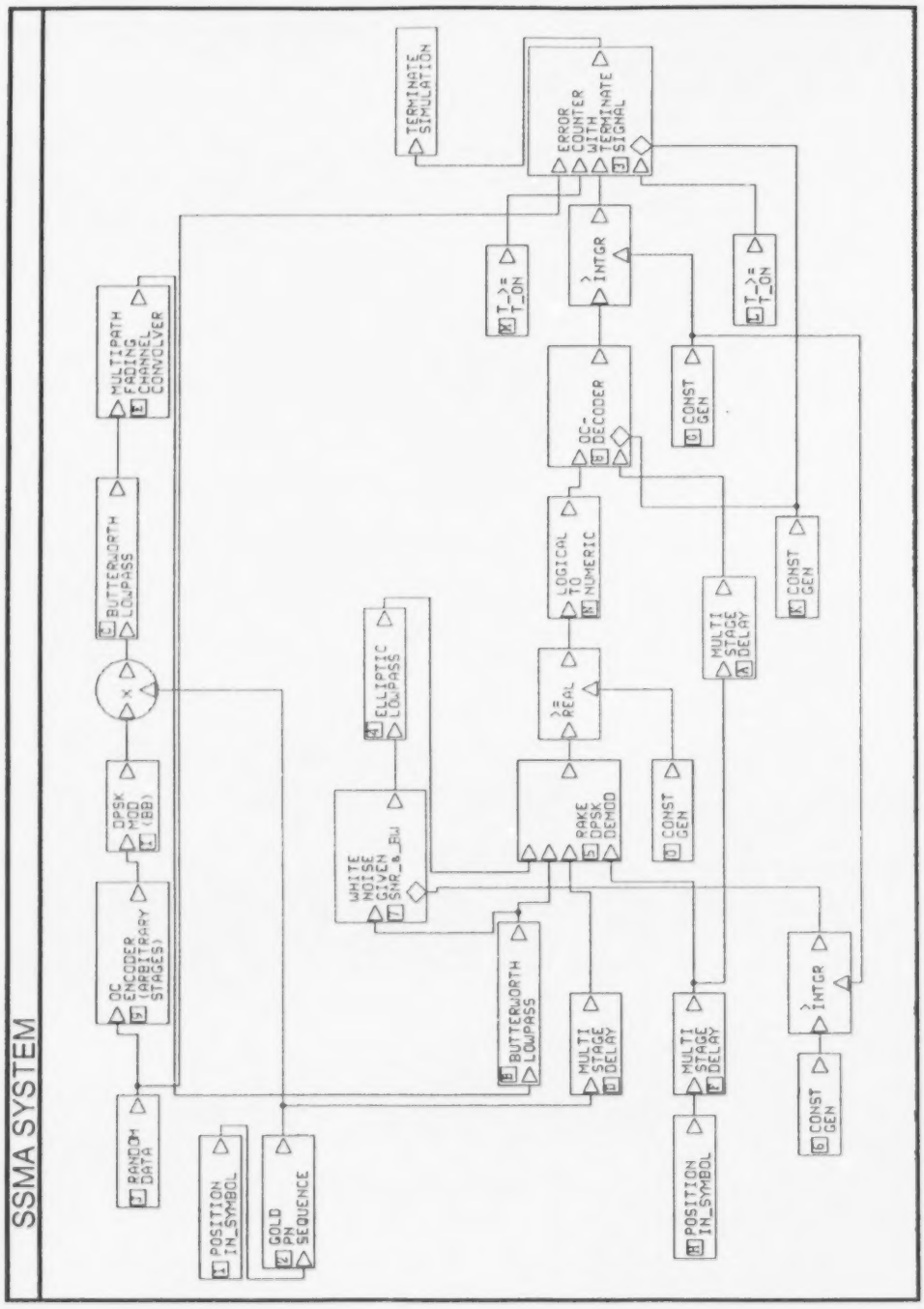


Figure A13. The coded SSMA system.

MODULE NAME: SSMA SYSTEM
GROUP: SYSTEM
DATABASE: /home/suns/rtse/model_db/
AUTHOR: rtse
CREATION DATE: 27-Aug-1992 21:42:03

DESCRIPTION:

A noncoherent test system for simulating SSMA on indoor multipath channels with orthogonal convolutional coding. The simulation processes baseband signals.

REVISIONS:

Author : rtse
Date : 27-Aug-1992 21:45:08
Description:

Edited 27-Aug-1992 21:45:08, No Edit Description Entered.

Author : rtse
Date : 27-Aug-1992 21:42:03
Description:

27-Aug-1992 21:42:10
Module CREATION.

INPUT SIGNALS:
(none)

OUTPUT SIGNALS:
(none)

PARAMETERS:

STOP-TIME Type: REAL
Lower Limit: 3.0E-39
Upper Limit: 1.7E38

Specifes the maximum value of time for the simulation (in seconds). The simulation clock runs from time=0.0 to time=STOP-TIME in steps of DT, all in seconds.

DT Type: REAL
Lower Limit: 3.0E-39
Upper Limit: 1.7E38

Time between discrete simulation signal samples (in seconds). DT must be small enough to satisfy the Nyquist Sampling

Theorem for all signals at all points in the simulation.

NOTE:
If the specified period or rate of a periodic function results in a period that is not a multiple of DT, then BOSS will round the period to the nearest value that IS a multiple of DT.
For example, if DT=0.125(sec) and a rate was specified as rate=1.4(hz) which corresponds to a period of T=0.714(sec) which is 5.7*DT , then BOSS will round this period to be 6.0*DT and thus the effective rate will be 1.333(hz).

Because of this, the choice for DT can affect the periods of the periodic signals in the simulation.

FILE NUMBER Type: INTEGER
Lower Limit: 1
Upper Limit: 20

The value entered here determines which file the output will be written to. Valid entries are integers from 1 to 20 inclusive. The output file is 'ber##.file' where ## represents the parameter value.

CHANNEL NUMBER Type: INTEGER
Lower Limit: 1
Upper Limit: 50

The channel file to read from to get the impulse response.

CHANNEL START NUMBER Type: INTEGER
Lower Limit: 1
Upper Limit: 19

This value determines which impulse response within a channel impulse response file to start at.

CODE CONSTRAINT LENGTH Type: INTEGER
Lower Limit: 1
Upper Limit: 10

The constraint length of the orthogonal convolutional code. This value is limited to a maximum of 10.

PATH MEMORY Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

The truncation length of the decoder. This value should equal 5 times the code constraint length for minimal loss of decoding gain.

DATA RATE Type: REAL
Lower Limit: 3.0E-39
Upper Limit: 1.7E38

The rate (in bits/sec) of the original uncoded data to be transmitted.

CODED DATA RATE Type: REAL
Lower Limit: 3.0E-39
Upper Limit: 1.7E38

The rate of the orthogonal convolutionally encoded data.

CHIP RATE Type: REAL
Lower Limit: 3.0E-39
Upper Limit: 1.7E38

The rate of the pn sequence, in bits/sec.

#ERRORS TO TERMINATE SIMULATION Type: INTEGER
Lower Limit: 0
Upper Limit: 2147483647

When this number of errors is detected, the terminate simulation signal output will go .true..

#BITS FOR A FILE UPDATE Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

The number of bits that are compared before the file inclusive).

DATA ISEED Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

The initial seed for the random data generator.

INT/NOISE ISEED1 Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

The initial seed used to generate the real part of thee noise/interference.

INT/NOISE ISEED2 Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

The initial seed used to generate the imaginary part of the noise/interference.

PN SEQUENCE NUMBER Type: INTEGER
Lower Limit: 1
Upper Limit: 1024

This number determines which pn code to use.

PHASE OFFSET Type: REAL
Lower Limit: -1.7E38
Upper Limit: 1.7E38

The absolute phase offset of the equivalent DPSK carrier waveform (in radians).

INTERFERERS Type: INTEGER
Lower Limit: 0
Upper Limit: 2147483647

This number determines the variance due to the number of interferers.

DIVERSITY ORDER Type: INTEGER
Lower Limit: 1
Upper Limit: 3

This integer determines the number of demodulators/decoders to use to recover the transmitted data. The minimum diversity order is 1 and the maximum is 3.

DECODER ENABLE Type: LOGICAL
Lower Limit: NIL
Upper Limit: NIL

If this value is .true., then the decoder will be enabled.
If it is .false., then the decoder will be disabled.

PN SEQUENCE LENGTH Type: INTEGER
Lower Limit: 1
Upper Limit: 5000

The number of symbols in the pn sequence.

CHANNEL VECTOR LENGTH Type: INTEGER
Lower Limit: 1
Upper Limit: 65

The number of samples in the multipath channel impulse response.

UPDATE PERIOD Type: REAL
Lower Limit: 3.0E-39
Upper Limit: 1.7E38

This is the time period between changes in the impulse response. That is, after each period equal to the entered value, a new impulse response will be read from the file.

The entered value should be an integer multiple of 'dt'.

COMPUTED PARAMETERS:

SNR Type: REAL
Lower Limit: -1.7E38
Upper Limit: 1.7E38

The signal to noise ratio after despreading. This is calculated by:

SNR = 10*log(2/(avg_xcorr*#interferers))

where avg_xcorr = mean cross correlation between partial or whole Gold sequences

```
   --------

   RX DELAY                                          Type: INTEGER
   Lower Limit: 1
   Upper Limit: 10000

     Number of samples (units) to delay the input signal. NOTE, minimum value
   --------

MODULES USED IN BLOCK DIAGRAM:
   >= REAL
   LOGICAL TO NUMERIC
   T_>= T_ON
   RANDOM DATA
   DPSK MOD (BB)
   MULTIPATH FADING CHANNEL CONVOLVER
   MULTIPLIER
   BUTTERWORTH LOWPASS
   MULTI STAGE DELAY
   OC ENCODER (ARBITRARY STAGES)
   OC-DECODER
   WHITE NOISE GIVEN SNR_&_BW
   CONST GEN
   > INTGR
   RAKE DPSK DEMOD
   ELLIPTIC LOWPASS
   TERMINATE SIMULATION
   ERROR COUNTER WITH TERMINATE SIGNAL
   GOLD PN SEQUENCE
   POSITION IN_SYMBOL

PARAMETER VALUES FOR INSTANCES IN BLOCK DIAGRAM:

POSITION IN_SYMBOL (key 1)
   TIME DELAY TO INPUT (SEC) == 0.0
   SYMBOL FRAC FOR SAMPLE TIME == 0.0
   SYMBOL RATE (HZ) == $CHIP RATE

GOLD PN SEQUENCE (key 2)
   USER NUMBER == $PN SEQUENCE NUMBER
   SIZE == $PN SEQUENCE LENGTH

ERROR COUNTER WITH TERMINATE SIGNAL (key 3)
   FILE NUMBER == $FILE NUMBER
   #ERRORS TO TERMINATE SIMULATION == $#ERRORS TO TERMINATE SIMULATION
   #BITS FOR A FILE UPDATE == $#BITS FOR A FILE UPDATE
   DATA RATE == $DATA RATE
   MAX DELAY (#BITS) == 100

ELLIPTIC LOWPASS (key 4)
   FILTER ORDER == 5
   PASSBAND EDGE (HZ) == $CODED DATA RATE
   PASSBAND RIPPLE (DB) == 0.5
   STOPBAND EDGE (HZ) == 1.1 * 'CODED DATA RATE'

RAKE DPSK DEMOD (key 5)
   CHIP RATE == $CHIP RATE
```

```
   DIVERSITY ORDER == $DIVERSITY ORDER

CONST GEN (key 6)
   CONSTANT VALUE == $# INTERFERERS

WHITE NOISE GIVEN SNR & BW (key 7)
   SIGNAL AVG PWR START TIME == 0
   SNR (DB) == $SNR
   NOISE BW == $CODED DATA RATE
   NOISE ON-TIME == 0
   REAL SEED == $INT/NOISE ISEED1
   IMAGINARY SEED == $INT/NOISE ISEED2

OC-DECODER (key 8)
   PATH MEMORY == $PATH MEMORY
   K == $CODE CONSTRAINT LENGTH
   VALUE OF 2^(2*K) ==  Round (2.0^(2.0 * 'CODE CONSTRAINT LENGTH'))
   VALUE OF 2^K ==  Round (2.0^'CODE CONSTRAINT LENGTH')

OC ENCODER (ARBITRARY STAGES) (key 9)
   DATA RATE == $DATA RATE
   CODE CONSTRAINT LENGTH == $CODE CONSTRAINT LENGTH

MULTI STAGE DELAY (key A)
   DELAY NUM ==  Round (2.0 / ('CHIP RATE' * 'DT'))

BUTTERWORTH LOWPASS (key B)
   FILTER ORDER == 3
   PASSBAND EDGE (HZ) == 0.5 * 'CHIP RATE'
   PASSBAND EDGE ATTENUATION (DB) == 3

BUTTERWORTH LOWPASS (key C)
   FILTER ORDER == 3
   PASSBAND EDGE (HZ) == 0.5 * 'CHIP RATE'
   PASSBAND EDGE ATTENUATION (DB) == 3

MULTI STAGE DELAY (key D)
   DELAY NUM == $RX DELAY

MULTIPATH FADING CHANNEL CONVOLVER (key E)
   CHANNEL NUMBER == $CHANNEL NUMBER
   CHANNEL START NUMBER == $CHANNEL START NUMBER
   UPDATE PERIOD == $UPDATE PERIOD
   VECTOR LENGTH == $CHANNEL VECTOR LENGTH

MULTI STAGE DELAY (key F)
   DELAY NUM == $RX DELAY

CONST GEN (key G)
   CONSTANT VALUE == 0

POSITION IN SYMBOL (key H)
   TIME DELAY TO INPUT (SEC) == 0.0
   SYMBOL FRAC FOR SAMPLE TIME == 1.0
   SYMBOL RATE (HZ) == $CODED DATA RATE

DPSK MOD (BB) (key I)
```

```
   BAUD RATE == $CODED DATA RATE
   PHASE OFFSET == $PHASE OFFSET

RANDOM DATA (key J)
   ISEED == $DATA ISEED
   PROBABILITY OF FALSE == 0.5
   BIT RATE == $DATA RATE

CONST GEN (key K)
   CONSTANT VALUE == $DECODER ENABLE

T_>= T_ON (key L)
   T-ON == ('PATH MEMORY' + 2.5) / 'DATA RATE'

T_>= T_ON (key M)
   T-ON == 0.5 / 'DATA RATE'

LOGICAL TO NUMERIC (key N)
   TRUE VALUE == -1
   FALSE VALUE == 1

CONST GEN (key O)
   CONSTANT VALUE == 0.0

INITIALIZATION CODE:


   Subroutine: zizSSMASYSTEMFFPBIEOK
   Arguments:
      FILE NUMBER
      # INTERFERERS
      CHANNEL NUMBER
      CHANNEL START NUMBER
      DIVERSITY ORDER
      CODED DATA RATE
      CHIP RATE
      CHANNEL VECTOR LENGTH
      SNR
      RX DELAY

   Description:

   The initialization code finds the optimum delay between
   the transmitter section and the receiver section for
   sampling.
   Also, the noise level due to the multi-user interference
   is calculated by the initialization code.
```

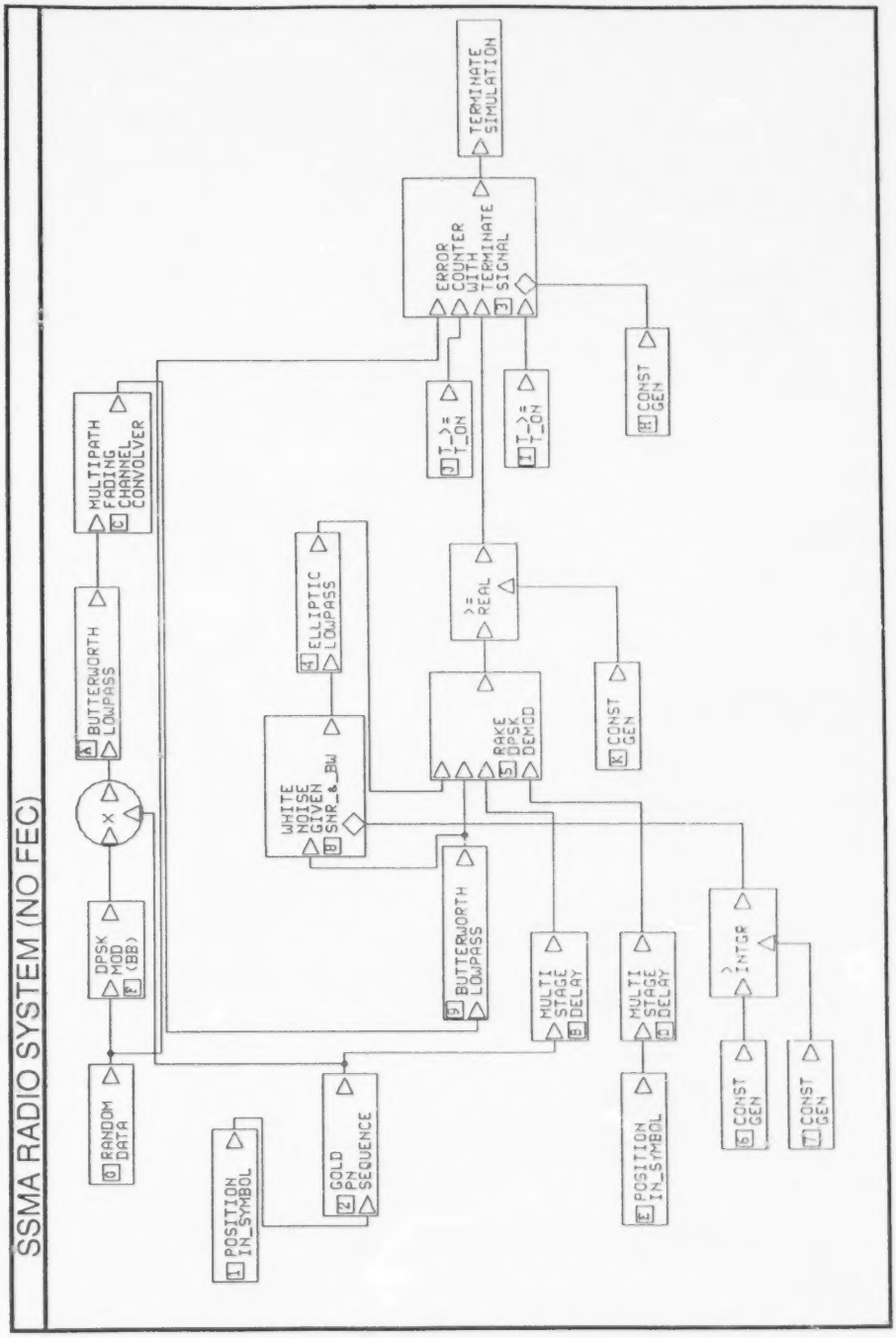


Figure A14. The uncoded SSMA system.

MODULE NAME: SSMA RADIO SYSTEM (NO FEC)
GROUP: SYSTEM
DATABASE: /home/suns/rtse/model_db/
AUTHOR: rtse
CREATION DATE: 25-Mar-1992 10:50:23

DESCRIPTION:

A noncoherent test system for simulating SSMA on indoor multipath channels. No FEC coding is used. The simulation processes baseband signals.

REVISIONS:

Author : rtse
Date : 13-Apr-1992 21:41:31
Description:

Edited 13-Apr-1992 21:41:31, No Edit Description Entered.

Author : rtse
Date : 13-Apr-1992 20:48:37
Description:

Edited 13-Apr-1992 20:48:37, No Edit Description Entered.

Author : rtse
Date : 25-Mar-1992 10:50:23
Description:

25-Mar-1992 10:50:24
Module CREATION.

INPUT SIGNALS:
(none)

OUTPUT SIGNALS:
(none)

PARAMETERS:

STOP-TIME Type: REAL
Lower Limit: 3.0E-39
Upper Limit: 1.7E38

Specifes the maximum value of time for the simulation (in seconds). The simulation clock runs from time=0.0 to time=STOP-TIME in steps of DT, all in seconds.

DT Type: REAL

Lower Limit: 3.0E-39
Upper Limit: 1.7E38

Time between discrete simulation signal samples (in seconds). DT must be small enough to satisfy the Nyquist Sampling Theorem for all signals at all points in the simulation.

NOTE:
If the specified period or rate of a periodic function results in a period that is not a multiple of DT, then BOSS will round the period to the nearest value that IS a multiple of DT.
For example, if DT=0.125(sec) and a rate was specified as rate=1.4(hz) which corresponds to a period of T=0.714(sec) which is 5.7*DT , then BOSS will round this period to be 6.0*DT and thus the effective rate will be 1.333(hz).

Because of this, the choice for DT can affect the periods of the periodic signals in the simulation.

FILE NUMBER Type: INTEGER
Lower Limit: 1
Upper Limit: 20

The value entered here determines which file the output will be written to. Valid entries are integers from 1 to 20 inclusive. The output file is 'ber##.file' where ## represents the parameter value.

CHANNEL NUMBER Type: INTEGER
Lower Limit: 1
Upper Limit: 50

The channel file to read from to get the impulse response.

CHANNEL START NUMBER Type: INTEGER
Lower Limit: 1
Upper Limit: 19

This value determines which impulse response within a channel impulse response file to start at.

DATA RATE Type: REAL
Lower Limit: 3.0E-39
Upper Limit: 1.7E38

The rate (in bits/sec) of the original uncoded data to be transmitted.

CHIP RATE Type: REAL
Lower Limit: 3.0E-39
Upper Limit: 1.7E38

The rate of the pn sequence, in bits/sec.

#ERRORS TO TERMINATE SIMULATION Type: INTEGER
Lower Limit: 0
Upper Limit: 2147483647

When this number of errors is detected, the terminate simulation signal output will go .true..

#BITS FOR A FILE UPDATE Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

The number of bits that are compared before the file
inclusive).

DATA ISEED Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

The initial seed for the random data generator.

INT/NOISE ISEED1 Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

initial seed used to generate the real part of thee
noise/interference.

INT/NOISE ISEED2 Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

The initial seed used to generate the imaginary part of the noise/interference.

PN SEQUENCE NUMBER Type: INTEGER

Lower Limit: 1
Upper Limit: 1024

This number determines which pn code to use.

PHASE OFFSET Type: REAL
Lower Limit: -1.7E38
Upper Limit: 1.7E38

The absolute phase offset of the equivalent DPSK carrier waveform (in radians).

INTERFERERS Type: INTEGER
Lower Limit: 0
Upper Limit: 2147483647

This number determines the variance due to the number of interferers.

DIVERSITY ORDER Type: INTEGER
Lower Limit: 1
Upper Limit: 3

This integer determines the number of demodulators/decoders to use to recover the transmitted data. The minimum diversity order is 1 and the maximum is 3.

UPDATE PERIOD Type: REAL
Lower Limit: 3.0E-39
Upper Limit: 1.7E38

This is the time period between changes in the impulse response. That is, after each period equal to the entered value, a new impulse response will be read from the file.

The entered value should be an integer multiple of 'dt'.

DECODER ENABLE Type: LOGICAL
Lower Limit: NIL
Upper Limit: NIL

If this value is .true., then the decoder will be enabled. If it is .false., then the decoder will be disabled.

PN SEQUENCE LENGTH Type: INTEGER
Lower Limit: 1
Upper Limit: 5000

The number of symbols in the pn sequence.

CHANNEL VECTOR LENGTH Type: INTEGER
Lower Limit: 1
Upper Limit: 65

The number of samples in the multipath channel impulse response.

COMPUTED PARAMETERS:

SNR Type: REAL
Lower Limit: -1.7E38
Upper Limit: 1.7E38

The signal to noise ratio after despreading. This value is calculated by:

```
snr = 10*log10(2.0/(avg_xcorr*#inter))
```

where avg_xcorr is the mean cross-correlation of partial or whole Gold sequences

---

RX DELAY Type: INTEGER
Lower Limit: 1
Upper Limit: 10000

Number of samples (units) to delay the input signal. NOTE, minimum value

---

MODULES USED IN BLOCK DIAGRAM:
>= REAL
T_>= T_ON
RANDOM DATA
DPSK MOD (BB)
MULTIPATH FADING CHANNEL CONVOLVER
MULTIPLIER
MULTI STAGE DELAY
BUTTERWORTH LOWPASS
WHITE NOISE GIVEN SNR_&_BW
> INTGR
CONST GEN
RAKE DPSK DEMOD

```
   ELLIPTIC LOWPASS
   TERMINATE SIMULATION
   ERROR COUNTER WITH TERMINATE SIGNAL
   GOLD PN SEQUENCE
   POSITION IN_SYMBOL

PARAMETER VALUES FOR INSTANCES IN BLOCK DIAGRAM:

POSITION IN_SYMBOL (key 1)
   TIME DELAY TO INPUT (SEC) == 0.0
   SYMBOL FRAC FOR SAMPLE TIME == 0.0
   SYMBOL RATE (HZ) == $CHIP RATE

GOLD PN SEQUENCE (key 2)
   USER NUMBER == $PN SEQUENCE NUMBER
   SIZE == $PN SEQUENCE LENGTH

ERROR COUNTER WITH TERMINATE SIGNAL (key 3)
   FILE NUMBER == $FILE NUMBER
   #ERRORS TO TERMINATE SIMULATION == $#ERRORS TO TERMINATE SIMULATION
   #BITS FOR A FILE UPDATE == $#BITS FOR A FILE UPDATE
   DATA RATE == $DATA RATE
   MAX DELAY (#BITS) == 100

ELLIPTIC LOWPASS (key 4)
   FILTER ORDER == 5
   PASSBAND EDGE (HZ) == $DATA RATE
   PASSBAND RIPPLE (DB) == 0.5
   STOPBAND EDGE (HZ) == 1.1 * 'DATA RATE'

RAKE DPSK DEMOD (key 5)
   CHIP RATE == $CHIP RATE
   DIVERSITY ORDER == $DIVERSITY ORDER

CONST GEN (key 6)
   CONSTANT VALUE == $# INTERFERERS

CONST GEN (key 7)
   CONSTANT VALUE == 0

WHITE NOISE GIVEN SNR_&_BW (key 8)
   SIGNAL AVG PWR START TIME == 0
   SNR (DB) == $SNR
   NOISE BW == $DATA RATE
   NOISE ON-TIME == 0
   REAL SEED == $INT/NOISE ISEED1
   IMAGINARY SEED == $INT/NOISE ISEED2

BUTTERWORTH LOWPASS (key 9)
   FILTER ORDER == 3
   PASSBAND EDGE (HZ) == 0.5 * 'CHIP RATE'
   PASSBAND EDGE ATTENUATION (DB) == 3

BUTTERWORTH LOWPASS (key A)
   FILTER ORDER == 3
   PASSBAND EDGE (HZ) == 0.5 * 'CHIP RATE'
   PASSBAND EDGE ATTENUATION (DB) == 3
```

```
MULTI STAGE DELAY (key B)
   DELAY NUM == $RX DELAY

MULTIPATH FADING CHANNEL CONVOLVER (key C)
   CHANNEL NUMBER == $CHANNEL NUMBER
   CHANNEL START NUMBER == $CHANNEL START NUMBER
   UPDATE PERIOD == $UPDATE PERIOD
   VECTOR LENGTH == $CHANNEL VECTOR LENGTH

MULTI STAGE DELAY (key D)
   DELAY NUM == $RX DELAY

POSITION IN SYMBOL (key E)
   TIME DELAY TO INPUT (SEC) == 0.0
   SYMBOL FRAC FOR SAMPLE TIME == 1.0
   SYMBOL RATE (HZ) == $DATA RATE

DPSK MOD (BB) (key F)
   BAUD RATE == $DATA RATE
   PHASE OFFSET == $PHASE OFFSET

RANDOM DATA (key G)
   ISEED == $DATA ISEED
   PROBABILITY OF FALSE == 0.5
   BIT RATE == $DATA RATE

CONST GEN (key H)
   CONSTANT VALUE == $DECODER ENABLE

T_>= T_ON (key I)
   T-ON == 2.5 / 'DATA RATE'

T_>= T_ON (key J)
   T-ON == 1.5 / 'DATA RATE'

CONST GEN (key K)
   CONSTANT VALUE == 0.0

INITIALIZATION CODE:


   Subroutine: zizSSMARADIOSYSTEELLDLHNK
   Arguments:
      FILE NUMBER
      # INTERFERERS
      CHANNEL NUMBER
      CHANNEL START NUMBER
      DIVERSITY ORDER
      DATA RATE
      CHIP RATE
      CHANNEL VECTOR LENGTH
      SNR
      RX DELAY

   Description:
```

The initialization code finds the number of samples of delay between the signal just before the transmit low-pass filter and the optimum sampling point at the receiver. Also, the code finds the SNR after despreading which results from the multi-user interference.

```
C                              ROUTINE_SYSTEM_INIT

C     This initialization code finds the appropriate delay required
C     in order to sample the signal at the optimum time and calcu-
C     lates the SNR after despreading of whole or partial Gold se-
C     quences.  The delay accounts for the multipath channel and
C     the low-pass filters **.  The number of samples per symbol
C     time, the diversity order of the RAKE (equal gain) receiver,
C     the channel length (in # of samples), the channel set to use,
C     and the impulse response within the set to be used need to be
C     specified.  The units of the output 'delay' is in # of samples.

C     ****************************************************************
C     The SNR of the system after despreading is calculated by the
C     following algorithm.

C   SNR = 10*log10(2.0/(avg_xcorr*#interferers))dB if #interferers > 0
C       = 30dB                                     if #interfersrs = 0
C
C     where avg_xcorr = the mean cross correlation between partial
C                       or whole Gold sequences

C     ****************************************************************

C     An extra 6 samples of delay is added to the output delay value
C     to account for the transmit and receive 3rd order Butterworth
C     low pass filter delay with a cut-off frequency equal to 16 MHz.
C     One sample is then subtracted to account for the fact that this
C     program calculates assuming a minimum delay of 1 where BOSS
C     assumes a minimum delay of 0.

      subroutine sample(file_num,inter,chnlnum,chlstrt,diver,
     &                  coded_rate,chip_rate,vec_len,snr,rx_delay)

      implicit none
      integer lun,no1,chlstrt,chnlnum,i,j,rx_delay,k,inter
      integer vec_len,diver,samp_sym,count,next,file_num,spread
      complex ch(80),sum(80),cmplx,out(80)
      real dist,time,mag,max,chip_rate,avg_xcorr
      real coded_rate,snr

      include '/bossdir/system/BOSSFORTRAN.INC'

      samp_sym = int(1.0/(chip_rate*dt)+0.5)

C     Read the channel impulse response
        call libget_lun(lun)
        no1 = lun
        go to(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,
     &        21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,
     &        37,38,39,40,41,42,43,44,45,46,47,48,49,50),chnlnum
    1   open(unit=no1,file='fc1a',status='old')
        go to 99
    2   open(unit=no1,file='fc2a',status='old')
        go to 99
    3   open(unit=no1,file='fc3a',status='old')
        go to 99
    4   open(unit=no1,file='fc4a',status='old')
        go to 99
    5   open(unit=no1,file='fc5a',status='old')
```

```
      go to 99
 6    open(unit=no1,file='fc6a',status='old')
      go to 99
 7    open(unit=no1,file='fc7a',status='old')
      go to 99
 8    open(unit=no1,file='fc8a',status='old')
      go to 99
 9    open(unit=no1,file='fc9a',status='old')
      go to 99
10    open(unit=no1,file='fc10a',status='old')
      go to 99
11    open(unit=no1,file='fc11a',status='old')
      go to 99
12    open(unit=no1,file='fc12a',status='old')
      go to 99
13    open(unit=no1,file='fc13a',status='old')
      go to 99
14    open(unit=no1,file='fc14a',status='old')
      go to 99
15    open(unit=no1,file='fc15a',status='old')
      go to 99
16    open(unit=no1,file='fc16a',status='old')
      go to 99
17    open(unit=no1,file='fc17a',status='old')
      go to 99
18    open(unit=no1,file='fc18a',status='old')
      go to 99
19    open(unit=no1,file='fc19a',status='old')
      go to 99
20    open(unit=no1,file='fc20a',status='old')
      go to 99
21    open(unit=no1,file='fc21a',status='old')
      go to 99
22    open(unit=no1,file='fc22a',status='old')
      go to 99
23    open(unit=no1,file='fc23a',status='old')
      go to 99
24    open(unit=no1,file='fc24a',status='old')
      go to 99
25    open(unit=no1,file='fc25a',status='old')
      go to 99
26    open(unit=no1,file='fc26a',status='old')
      go to 99
27    open(unit=no1,file='fc27a',status='old')
      go to 99
28    open(unit=no1,file='fc28a',status='old')
      go to 99
29    open(unit=no1,file='fc29a',status='old')
      go to 99
30    open(unit=no1,file='fc30a',status='old')
      go to 99
31    open(unit=no1,file='fc31a',status='old')
      go to 99
32    open(unit=no1,file='fc32a',status='old')
      go to 99
33    open(unit=no1,file='fc33a',status='old')
      go to 99
34    open(unit=no1,file='fc34a',status='old')
      go to 99
35    open(unit=no1,file='fc35a',status='old')
```

```
      go to 99
 36   open(unit=no1,file='fc36a',status='old')
      go to 99
 37   open(unit=no1,file='fc37a',status='old')
      go to 99
 38   open(unit=no1,file='fc38a',status='old')
      go to 99
 39   open(unit=no1,file='fc39a',status='old')
      go to 99
 40   open(unit=no1,file='fc40a',status='old')
      go to 99
 41   open(unit=no1,file='fc41a',status='old')
      go to 99
 42   open(unit=no1,file='fc42a',status='old')
      go to 99
 43   open(unit=no1,file='fc43a',status='old')
      go to 99
 44   open(unit=no1,file='fc44a',status='old')
      go to 99
 45   open(unit=no1,file='fc45a',status='old')
      go to 99
 46   open(unit=no1,file='fc46a',status='old')
      go to 99
 47   open(unit=no1,file='fc47a',status='old')
      go to 99
 48   open(unit=no1,file='fc48a',status='old')
      go to 99
 49   open(unit=no1,file='fc49a',status='old')
      go to 99
 50   open(unit=no1,file='fc50a',status='old')
 99   continue
      do i = 1,chlstrt,1
        do k = 1,vec_len,1
          read(no1,100) dist,time,ch(k)
100       format(f10.6,f12.6,e14.6,e14.6)
        enddo
      enddo
      close(no1,status='keep')
      call libfree_lun(no1)

C     Find the number of sample delays required so that sampling will
C     be done at the optimum point.  If diversity is used, then this
C     is also taken into account (assuming equal gain combining).

C     Find output of square pulse convolved with channel profile.
      do i = 1,vec_len+samp_sym-1,1
        do j = 1,samp_sym,1
          if (i-j+1.ge.1) then
            if (i-j+1.le.vec_len) then
              out(i) = out(i) + ch(i-j+1)
            endif
          endif
        enddo
      enddo

      do count = 1,vec_len+samp_sym-1-samp_sym*diver,1
        do j = 1,diver,1
          next = (j-1)*samp_sym+count
          do i = next,next+samp_sym-1,1
            sum(count) = sum(count)+out(i)
```

```
            enddo
            mag = cabs(sum(count))**2.0+mag
            sum(count) = cmplx(0.0,0.0)
          enddo
          if (mag.gt.max) then
             max = mag
             rx_delay = count-1+6
             if (chip_rate.lt.30.0E6) then
                rx_delay = rx_delay+7
             endif
          endif
          mag = 0.0
        enddo

C       Get the correct value for avg_xcorr.
        spread = int(log(10.0)*log10(chip_rate/coded_rate)/
       &          log(2.0)+0.5)+1
        goto (1000,1001,1002,1003,1004,1005,1006,1007,1008,
       &      1009,1010) spread
 1000   avg_xcorr = 0.75/1.0
        goto 1011
 1001   avg_xcorr = 0.86321/2.0
        goto 1011
 1002   avg_xcorr = 1.26267/4.0
        goto 1011
 1003   avg_xcorr = 1.80442/8.0
        goto 1011
 1004   avg_xcorr = 2.57023/16.0
        goto 1011
 1005   avg_xcorr = 3.32039/32.0
        goto 1011
 1006   avg_xcorr = 5.05359/64.0
        goto 1011
 1007   avg_xcorr = 7.35673/128.0
        goto 1011
 1008   avg_xcorr = 10.39165/256.0
        goto 1011
 1009   avg_xcorr = 14.4094/512.0
        goto 1011
 1010   avg_xcorr = 19.7768/1024.0
 1011   continue

        snr = 10.0*log10(2.0/(avg_xcorr*inter))
        if (inter.eq.0) then
          snr = 30.0
        endif


C       This last part writes the delay and the resultant magnitude
C       to the file 'simdat.file'.
        call libget_lun(lun)
        open(unit=lun,file='siminfo.file',access='append')
        write(lun,*) 'file# =',file_num,'  channel =',chnlnum,
       &             '-',chlstrt
        write(lun,*) '  delay =',rx_delay,'  mag =',max
        write(lun,*) '  #interferers =',inter
        write(lun,*) '  despread snr =',snr,'dB'
        write(lun,*) ' '
        close(lun,status='keep')
        call libfree_lun(lun)
```

```
return
end
```

A1.ix SSMA System for Use with Error Bound

This system is identical to the one in A1.viii except that the decoder has been removed and the error counter has been moved to count the number of symbol errors rather than the bit errors. The initialization code used for this system is ROUTINE_SYSTEM_INIT. This is the same code as used for the system described in A1.viii.

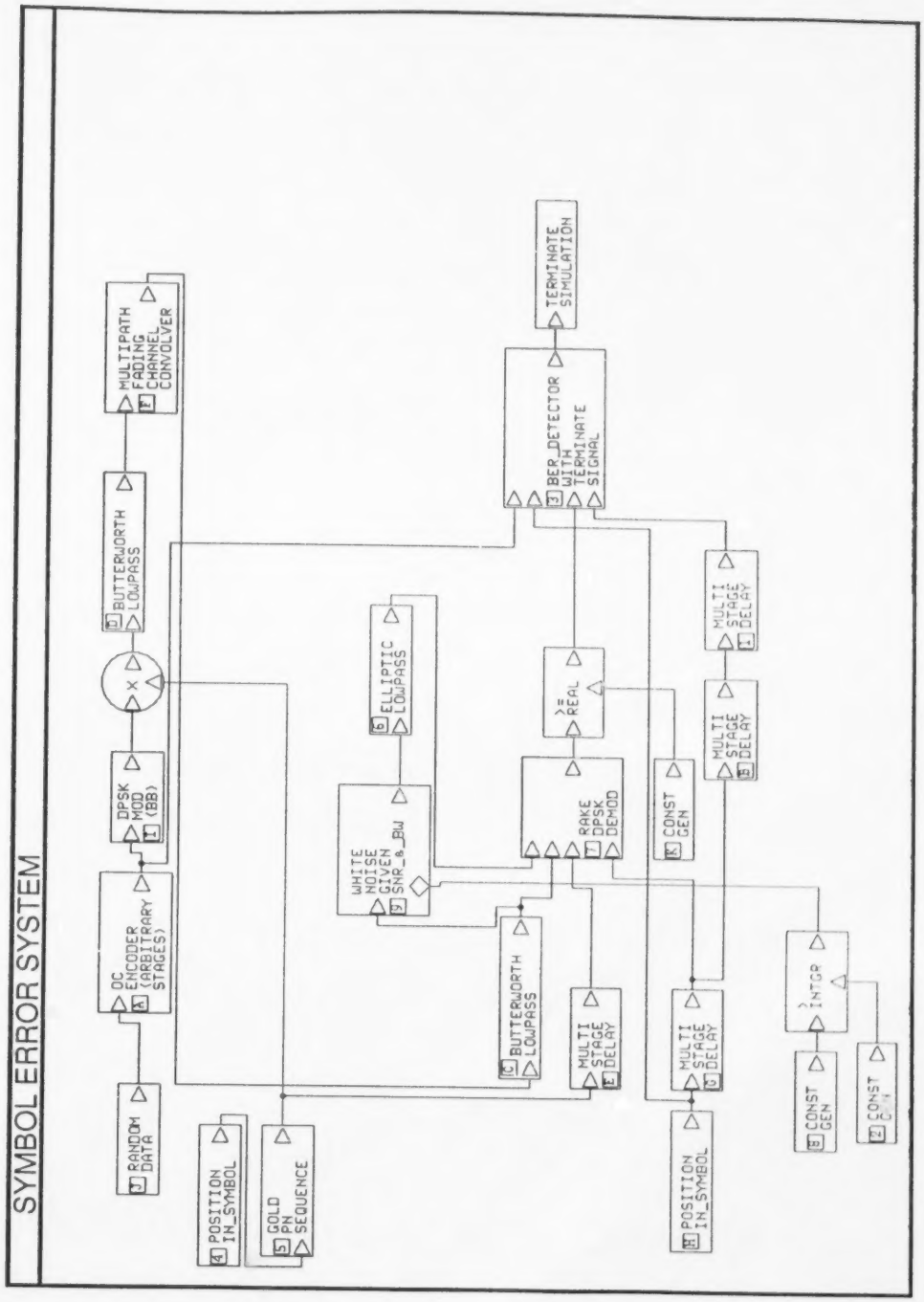


Figure A15. The SSMA system for use with the error bounding method.

MODULE NAME: SYMBOL ERROR SYSTEM
GROUP: SYSTEM
DATABASE: /home/suns/rtse/model_db/
AUTHOR: rtse
CREATION DATE: 23-Jul-1992 15:34:34

DESCRIPTION:

A noncoherent test system for simulating SSMA on indoor multipath channels with orthogonal convolutional coding. The simulation processes baseband signals and counts the encoded symbol error rate.

The results of the encoded symbol error rate are written to the file 'berxx.file', where xx is a number from 1 to 20 inclusive.

REVISIONS:

Author : rtse
Date : 30-Jul-1992 11:13:27
Description:

Edited 30-Jul-1992 11:13:27, No Edit Description Entered.

Author : rtse
Date : 23-Jul-1992 15:34:34
Description:

23-Jul-1992 15:34:36
Module CREATION.

INPUT SIGNALS:
(none)

OUTPUT SIGNALS:
(none)

PARAMETERS:

STOP-TIME Type: REAL
Lower Limit: 3.0E-39
Upper Limit: 1.7E38

Specifes the maximum value of time for the simulation (in seconds). The simulation clock runs from time=0.0 to time=STOP-TIME in steps of DT, all in seconds.

DT Type: REAL
Lower Limit: 3.0E-39

Upper Limit: 1.7E38

Time between discrete simulation signal samples (in seconds). DT must be small enough to satisfy the Nyquist Sampling Theorem for all signals at all points in the simulation.

NOTE:
If the specified period or rate of a periodic function results in a period that is not a multiple of DT, then BOSS will round the period to the nearest value that IS a multiple of DT.
For example, if DT=0.125(sec) and a rate was specified as rate=1.4(hz) which corresponds to a period of T=0.714(sec) which is 5.7*DT , then BOSS will round this period to be 6.0*DT and thus the effective rate will be 1.333(hz).

Because of this, the choice for DT can affect the periods of the periodic signals in the simulation.

FILE NUMBER Type: INTEGER
Lower Limit: 1
Upper Limit: 20

The value entered here determines which file the output will be written to. Valid entries are integers from 1 to 20 inclusive. The output file is 'ber##.file' where ## represents the parameter value.

CHANNEL NUMBER Type: INTEGER
Lower Limit: 1
Upper Limit: 50

The channel file to read from to get the impulse response.

CHANNEL START NUMBER Type: INTEGER
Lower Limit: 1
Upper Limit: 19

This value determines which impulse response within a channel impulse response file to start at.

CODE CONSTRAINT LENGTH Type: INTEGER
Lower Limit: 1
Upper Limit: 10

The constraint length of the orthogonal convolutional code. This value is limited to a maximum of 10.

DATA RATE Type: REAL
Lower Limit: 3.0E-39
Upper Limit: 1.7E38

The rate (in bits/sec) of the original uncoded data to be transmitted.

CODED DATA RATE Type: REAL
Lower Limit: 3.0E-39
Upper Limit: 1.7E38

The rate of the orthogonal convolutionally encoded data.

CHIP RATE Type: REAL
Lower Limit: 3.0E-39
Upper Limit: 1.7E38

The rate of the pn sequence, in bits/sec.

#ERRORS TO TERMINATE SIMULATION Type: INTEGER
Lower Limit: 0
Upper Limit: 2147483647

When this number of errors is detected, the terminate simulation signal output will go .true..

#BITS FOR A FILE UPDATE Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

The number of bits that are compared before the file inclusive).

DATA ISEED Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

The initial seed for the random data generator.

INT/NOISE ISEED1 Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

The initial seed used to generate the real part of thee noise/interference.

INT/NOISE ISEED2 Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

The initial seed used to generate the imaginary part of the noise/interference.

PN SEQUENCE NUMBER Type: INTEGER
Lower Limit: 1
Upper Limit: 1024

This number determines which pn code to use.

PHASE OFFSET Type: REAL
Lower Limit: -1.7E38
Upper Limit: 1.7E38

The absolute phase offset of the equivalent DPSK carrier waveform (in radians).

INTERFERERS Type: INTEGER
Lower Limit: 0
Upper Limit: 2147483647

This number determines the variance due to the number of interferers.

DIVERSITY ORDER Type: INTEGER
Lower Limit: 1
Upper Limit: 3

This integer determines the number of demodulators/decoders to use to recover the transmitted data. The minimum diversity order is 1 and the maximum is 3.

PN SEQUENCE LENGTH Type: INTEGER
Lower Limit: 1
Upper Limit: 5000

The number of symbols in the pn sequence.

CHANNEL VECTOR LENGTH Type: INTEGER
Lower Limit: 1
Upper Limit: 65

The number of samples in the multipath channel impulse response.

UPDATE PERIOD Type: REAL
Lower Limit: 3.0E-39
Upper Limit: 1.7E38

This is the time period between changes in the impulse response. That is, after each period equal to the entered value, a new impulse response will be read from the file.

The entered value should be an integer multiple of 'dt'.

COMPUTED PARAMETERS:

SNR Type: REAL
Lower Limit: -1.7E38
Upper Limit: 1.7E38

The signal to noise ratio after despreading. This is calculated by:

SNR = 10*log(2/(avg_xcorr*#interferers))

where avg_xcorr = mean cross correlation between partial or whole Gold sequences

---

RX DELAY Type: INTEGER
Lower Limit: 1
Upper Limit: 10000

Number of samples (units) to delay the input signal. NOTE, minimum value
---

MODULES USED IN BLOCK DIAGRAM:
>= REAL
RANDOM DATA
DPSK MOD (BB)

```
   MULTIPATH FADING CHANNEL CONVOLVER
   MULTIPLIER
   BUTTERWORTH LOWPASS
   OC ENCODER (ARBITRARY STAGES)
   WHITE NOISE GIVEN SNR_&_BW
   > INTGR
   RAKE DPSK DEMOD
   ELLIPTIC LOWPASS
   TERMINATE SIMULATION
   GOLD PN SEQUENCE
   POSITION IN_SYMBOL
   BER_DETECTOR WITH TERMINATE SIGNAL
   CONST GEN
   MULTI STAGE DELAY

PARAMETER VALUES FOR INSTANCES IN BLOCK DIAGRAM:

MULTI STAGE DELAY (key 1)
   DELAY NUM ==  Round (1.0 / ('CODED DATA RATE' * 'DT'))

CONST GEN (key 2)
   CONSTANT VALUE == 0

BER_DETECTOR WITH TERMINATE SIGNAL (key 3)
   #ERRORS TO TERMINATE SIMULATION == $#ERRORS TO TERMINATE SIMULATION
   FILE NUMBER == $FILE NUMBER
   #BITS FOR A FILE UPDATE == $#BITS FOR A FILE UPDATE
   MAX DELAY (#BITS) == 100

POSITION IN_SYMBOL (key 4)
   TIME DELAY TO INPUT (SEC) == 0.0
   SYMBOL FRAC FOR SAMPLE TIME == 0.0
   SYMBOL RATE (HZ) == $CHIP RATE

GOLD PN SEQUENCE (key 5)
   USER NUMBER == $PN SEQUENCE NUMBER
   SIZE == $PN SEQUENCE LENGTH

ELLIPTIC LOWPASS (key 6)
   FILTER ORDER == 5
   PASSBAND EDGE (HZ) == $CODED DATA RATE
   PASSBAND RIPPLE (DB) == 0.5
   STOPBAND EDGE (HZ) == 1.1 * 'CODED DATA RATE'

RAKE DPSK DEMOD (key 7)
   CHIP RATE == $CHIP RATE
   DIVERSITY ORDER == $DIVERSITY ORDER

CONST GEN (key 8)
   CONSTANT VALUE == $# INTERFERERS

WHITE NOISE GIVEN SNR_&_BW (key 9)
   SIGNAL AVG PWR START TIME == 0
   SNR (DB) == $SNR
   NOISE BW == $CODED DATA RATE
   NOISE ON-TIME == 0
   REAL SEED == $INT/NOISE ISEED1
```

```
    IMAGINARY SEED == $INT/NOISE ISEED2

OC ENCODER (ARBITRARY STAGES) (key A)
   DATA RATE == $DATA RATE
   CODE CONSTRAINT LENGTH == $CODE CONSTRAINT LENGTH

MULTI STAGE DELAY (key B)
   DELAY NUM ==  Round (2.0 / ('CHIP RATE' * 'DT'))

BUTTERWORTH LOWPASS (key C)
   FILTER ORDER == 3
   PASSBAND EDGE (HZ) == 0.5 * 'CHIP RATE'
   PASSBAND EDGE ATTENUATION (DB) == 3

BUTTERWORTH LOWPASS (key D)
   FILTER ORDER == 3
   PASSBAND EDGE (HZ) == 0.5 * 'CHIP RATE'
   PASSBAND EDGE ATTENUATION (DB) == 3

MULTI STAGE DELAY (key E)
   DELAY NUM == $RX DELAY

MULTIPATH FADING CHANNEL CONVOLVER (key F)
   CHANNEL NUMBER == $CHANNEL NUMBER
   CHANNEL START NUMBER == $CHANNEL START NUMBER
   UPDATE PERIOD == $UPDATE PERIOD
   VECTOR LENGTH == $CHANNEL VECTOR LENGTH

MULTI STAGE DELAY (key G)
   DELAY NUM == $RX DELAY

POSITION IN_SYMBOL (key H)
   TIME DELAY TO INPUT (SEC) == 0.0
   SYMBOL FRAC FOR SAMPLE TIME == 1.0
   SYMBOL RATE (HZ) == $CODED DATA RATE

DPSK MOD (BB) (key I)
   BAUD_RATE == $CODED DATA RATE
   PHASE OFFSET == $PHASE OFFSET

RANDOM DATA (key J)
   ISEED == $DATA ISEED
   PROBABILITY OF FALSE == 0.5
   BIT RATE == $DATA RATE

CONST GEN (key K)
   CONSTANT VALUE == 0.0

INITIALIZATION CODE:


   Subroutine: zizSYMBOLERRORSYSPEBKCCOK
   Arguments:
      FILE NUMBER
      # INTERFERERS
      CHANNEL NUMBER
      CHANNEL START NUMBER
```

DIVERSITY ORDER
CODED DATA RATE
CHIP RATE
CHANNEL VECTOR LENGTH
SNR
RX DELAY

Description:

The initialization code finds the optimum delay between the transmitter section and the receiver section for sampling.
Also, the noise level due to the multi-user interference is calculated by the initialization code.

A1.x Interference Generating System

Figure A16 shows the module which produces one interference signal for the interference generating system. The system is used to write samples of interference for statistical analysis. Each interference module generates random data at the chip rate to represent the spread signal at baseband. Each signal is subjected to a random phase offset, a random time delay (up to 1 chip time), and an activation factor of 0.5. The result is convolved through a multipath channel unique to each interferer.

Any number of interferers can be simulated by summing the outputs of an equivalent number of these modules. The summed signals are then low-pass filtered by two third-order Butterworth transmit and receive filters with 3 dB frequencies equal to half the chip rate. The resulting signal is then multiplied with a despreading sequence. The product can be analyzed for its approximation to white Gaussian noise.

An interference generating system for 9 interfering users is shown in Figure A17.

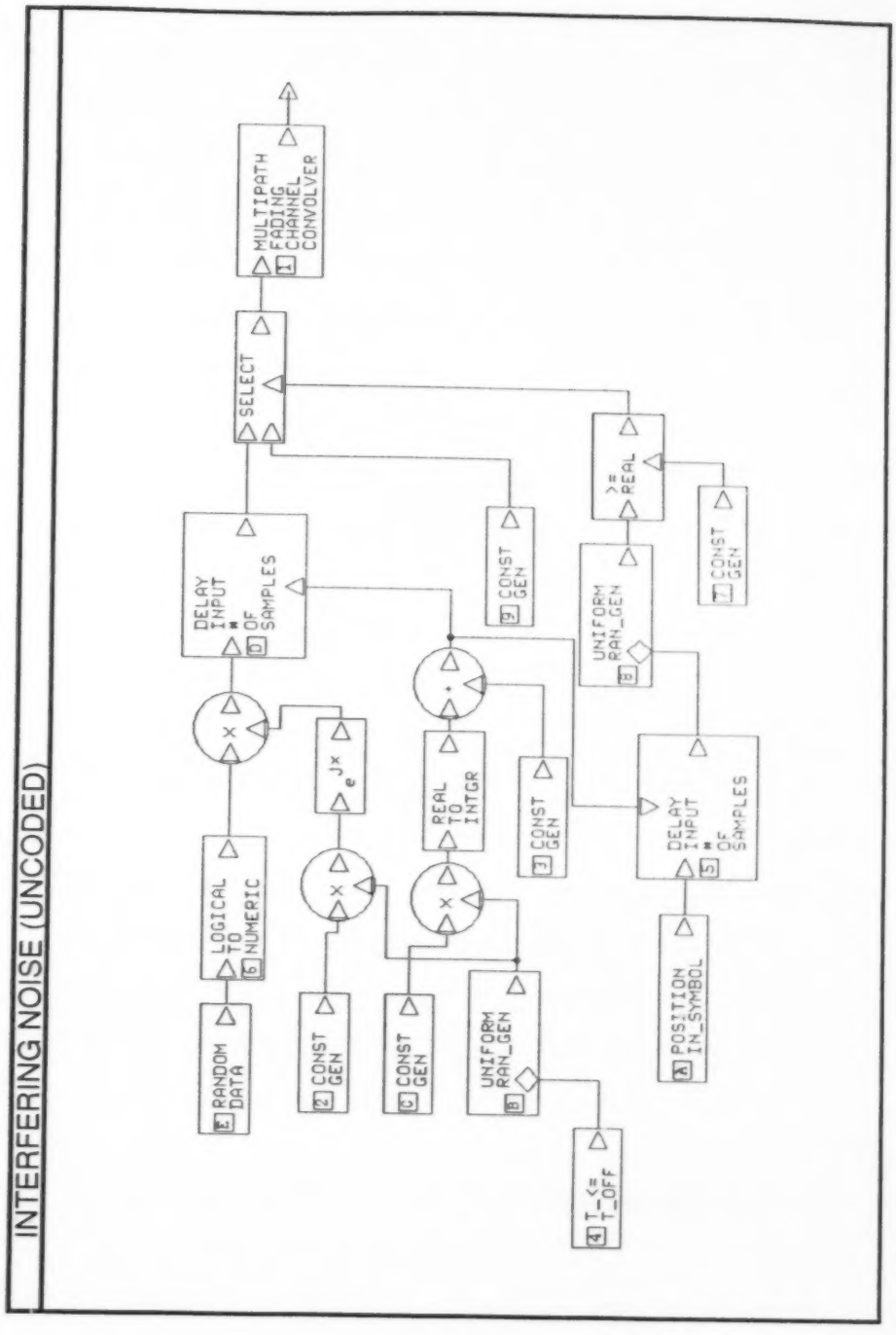


Figure A16. Interference generator.

MODULE NAME: INTERFERING NOISE (UNCODED)
GROUP: NOISE AND INTERFERENCE
DATABASE: /home/suns/rtse/model_db/
AUTHOR: rtse
CREATION DATE: 24-Oct-1991 20:42:55

DESCRIPTION:

This module generates the result of a random chip stream, REPRESENTING a FEC coded data stream with a 1/n rate orthogonal convolutional code which is then spread with a gold sequence, which is randomly delayed, transformed into an impulse, and then convolved with an indoor multipath fading channel.

REVISIONS:

Author : rtse
Date : 31-Jan-1992 13:29:05
Description:

Edited 31-Jan-1992 13:29:05, No Edit Description Entered.

Author : rtse
Date : 19-Dec-1991 14:11:18
Description:

Edited 19-Dec-1991 14:11:18, No Edit Description Entered.

Author : rtse
Date : 12-Dec-1991 16:50:50
Description:

Edited 12-Dec-1991 16:50:50, No Edit Description Entered.

Author : rtse
Date : 6-Dec-1991 15:04:12
Description:

Edited 3-Dec-1991 16:26:33, No Edit Description Entered.
Edited 6-Dec-1991 10:12:29, No Edit Description Entered.
Edited 6-Dec-1991 15:04:12, No Edit Description Entered.

Author : rtse
Date : 4-Nov-1991 14:40:46
Description:

Edited 4-Nov-1991 14:40:46, No Edit Description Entered.

Author : rtse
Date : 24-Oct-1991 20:42:55
Description:

24-Oct-1991 20:43:29
Module CREATION.

INPUT SIGNALS:
(none)

OUTPUT SIGNALS:

OUTPUT
Lower Limit: (-1.7E38 -1.7E38)
Upper Limit: (1.7E38 1.7E38)
Type: COMPLEX

The output of the interfering user. The result is a coded data signal impulse, randomly delayed, and passed through an indoor multipath channel.

PARAMETERS:

CHIP RATE
Lower Limit: 3.0E-39
Upper Limit: 1.7E38
Type: REAL

The chip rate of a DS-SSMA interfering user.

CHANNEL NUMBER
Lower Limit: 1
Upper Limit: 50
Type: INTEGER

The channel file to read from to get the impulse response.

CHANNEL START NUMBER
Lower Limit: 1
Upper Limit: 19
Type: INTEGER

This value determines which impulse response within a channel impulse response file to start at.

CHANNEL VECTOR LENGTH
Lower Limit: 1
Upper Limit: 65
Type: INTEGER

The number of samples in the multipath fading channel impulse response.

DATA SEED Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

The input seed for the random data generator.

DELAY SEED Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

The seed to generate the random number which determines the number of samples to delay the signal.

ACTIVATION SEED Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

The seed to generate a uniform random number for the activation.

UPDATE PERIOD Type: REAL
Lower Limit: 3.0E-39
Upper Limit: 1.7E38

This is the time period between changes in the impulse response. That is, after each period equal to the entered value, a new impulse response will be read from the file.

The entered value should be an integer multiple of 'dt'.

MODULES USED IN BLOCK DIAGRAM:
- RANDOM DATA
- REAL TO INTGR
- SELECT
- POSITION IN_SYMBOL
- UNIFORM RAN_GEN
- >= REAL
- LOGICAL TO NUMERIC
- DELAY INPUT # OF SAMPLES
- T_<= T_OFF
- ADDER
- CONST GEN
- CMPLX EXPONENTIAL
- MULTIPLIER
- MULTIPATH FADING CHANNEL CONVOLVER

```
PARAMETER VALUES FOR INSTANCES IN BLOCK DIAGRAM:

MULTIPATH FADING CHANNEL CONVOLVER (key 1)
   CHANNEL NUMBER == $CHANNEL NUMBER
   CHANNEL START NUMBER == $CHANNEL START NUMBER
   UPDATE PERIOD == $UPDATE PERIOD
   VECTOR LENGTH == $CHANNEL VECTOR LENGTH

CONST GEN (key 2)
   CONSTANT VALUE == 6.283185307179586

CONST GEN (key 3)
   CONSTANT VALUE == 1

T_<= T_OFF (key 4)
   T-OFF == 0.5 * 'DT'

DELAY INPUT # OF SAMPLES (key 5)
   MAX DELAY == 5000

LOGICAL TO NUMERIC (key 6)
   TRUE VALUE == (1.0 , 0.0)
   FALSE VALUE == (-1.0 , 0.0)

CONST GEN (key 7)
   CONSTANT VALUE == 0.5

UNIFORM RAN_GEN (key 8)
   ISEED == $ACTIVATION SEED

CONST GEN (key 9)
   CONSTANT VALUE == (0.0 , 0.0)

POSITION IN_SYMBOL (key A)
   TIME DELAY TO INPUT (SEC) == 0
   SYMBOL FRAC FOR SAMPLE TIME == 0
   SYMBOL RATE (HZ) == $CHIP RATE

UNIFORM RAN_GEN (key B)
   ISEED == $DELAY SEED

CONST GEN (key C)
   CONSTANT VALUE ==  Round (1.0 / ('CHIP RATE' * 'DT'))

DELAY INPUT # OF SAMPLES (key D)
   MAX DELAY == 5000

RANDOM DATA (key E)
   ISEED == $DATA SEED
   PROBABILITY OF FALSE == 0.5
   BIT RATE == $CHIP RATE

INITIALIZATION CODE:
   (none)
```

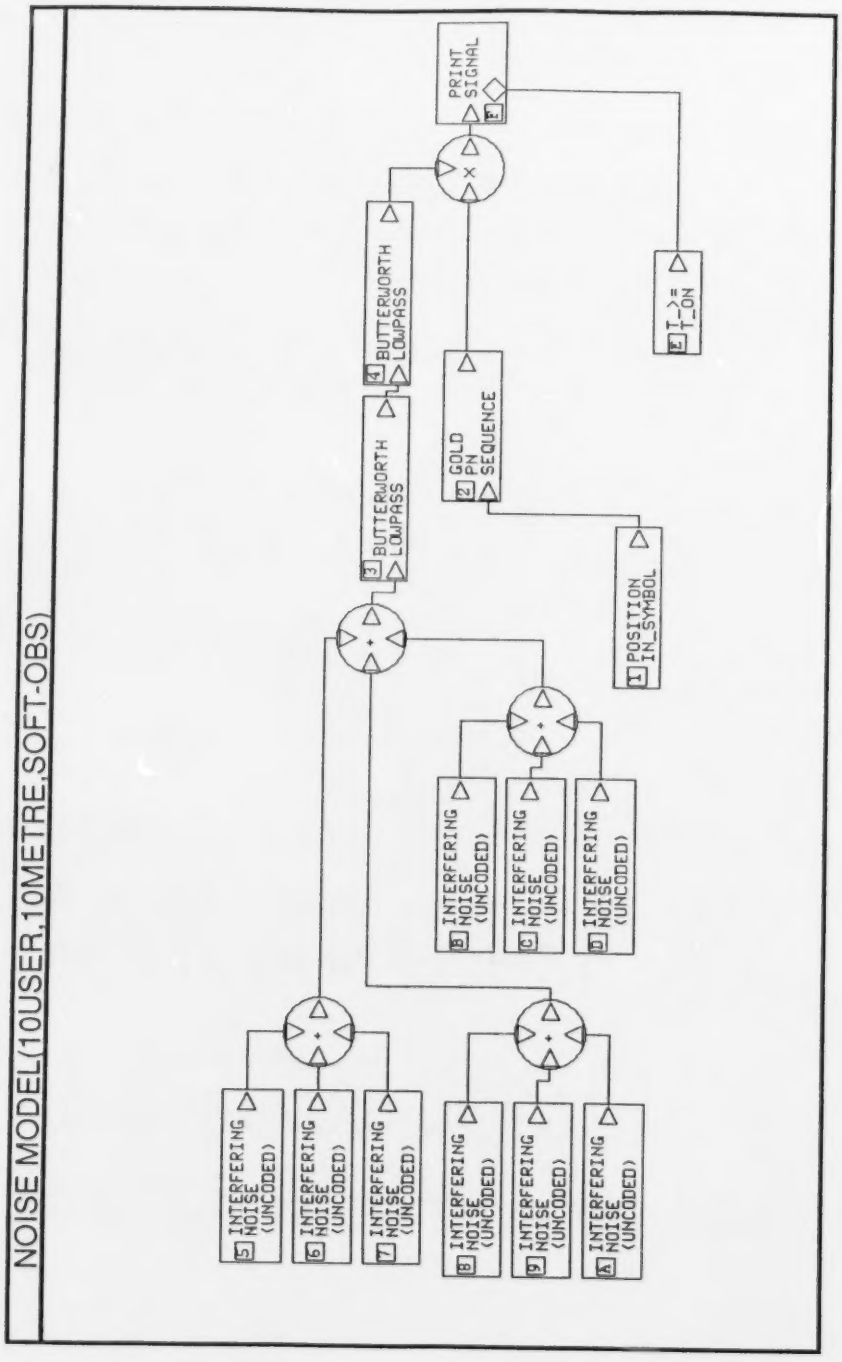


Figure A17. System for writing interference samples of 9 interferers.

MODULE NAME: NOISE MODEL(10USER,10METRE,SOFT-OBS)
GROUP: SYSTEM
DATABASE: /home/suns/rtse/model_db/
AUTHOR: rtse
CREATION DATE: 20-Jan-1992 14:22:48

DESCRIPTION:

This module generates the result of 9 random chip streams, representing signals coded with 1/n rate orthogonal convolutional codes and spread with gold sequences, which are randomly delayed, convolved with an indoor multipath channel, and then passed through a 3rd order butterworth low pass filter with passband equal to 0.5* the chip rate. The 'activation factor' is 0.5.

REVISIONS:

Author : rtse
Date : 4-May-1992 13:12:02
Description:

Edited 4-May-1992 13:12:02, No Edit Description Entered.

Author : rtse
Date : 4-May-1992 11:09:24
Description:

Edited 4-May-1992 11:09:24, No Edit Description Entered.

Author : rtse
Date : 4-Feb-1992 13:07:19
Description:

This module was MOVED into this data base.

Author : rtse
Date : 20-Jan-1992 14:22:48
Description:

20-Jan-1992 14:22:50
Module CREATION.

INPUT SIGNALS:
(none)

OUTPUT SIGNALS:
(none)

PARAMETERS:

STOP-TIME
Type: REAL
Lower Limit: 3.0E-39
Upper Limit: 1.7E38

Specifes the maximum value of time for the simulation (in seconds). The simulation clock runs from time=0.0 to time=STOP-TIME in steps of DT, all in seconds.

DT
Type: REAL
Lower Limit: 3.0E-39
Upper Limit: 1.7E38

Time between discrete simulation signal samples (in seconds). DT must be small enough to satisfy the Nyquist Sampling Theorem for all signals at all points in the simulation.

NOTE:
If the specified period or rate of a periodic function results in a period that is not a multiple of DT, then BOSS will round the period to the nearest value that IS a multiple of DT. For example, if DT=0.125(sec) and a rate was specified as rate=1.4(hz) which corresponds to a period of T=0.714(sec) which is 5.7*DT , then BOSS will round this period to be 6.0*DT and thus the effective rate will be 1.333(hz).

Because of this, the choice for DT can affect the periods of the periodic signals in the simulation.

USER NUMBER
Type: INTEGER
Lower Limit: 1
Upper Limit: 2147483647

The user number corresponds to the nth user in the multiple access system.

CHANNEL VECTOR LENGTH
Type: INTEGER
Lower Limit: 1
Upper Limit: 65

The number of samples in the multipath fading channel impulse response.

CHIP RATE
Type: REAL
Lower Limit: 3.0E-39
Upper Limit: 1.7E38

The chip rate of a DS-SSMA interfering user.

```
    --------

    UPDATE PERIOD                                          Type: REAL
    Lower Limit: 3.0E-39
    Upper Limit: 1.7E38

    This is the time period between changes in the impulse
    response. That is, after each period equal to the
    entered value, a new impulse response will be read from
    the file.

    The entered value should be an integer multiple of 'dt'.

    --------

    T-ON                                                   Type: REAL
    Lower Limit: 0.0
    Upper Limit: 1.7E38

      Time (seconds) in which enable becomes true.
    --------

MODULES USED IN BLOCK DIAGRAM:
   PRINT SIGNAL
   T_>= T_ON
   3_INPUT ADDER
   INTERFERING NOISE (UNCODED)
   BUTTERWORTH LOWPASS
   GOLD PN SEQUENCE
   POSITION IN_SYMBOL
   MULTIPLIER

PARAMETER VALUES FOR INSTANCES IN BLOCK DIAGRAM:

POSITION IN_SYMBOL (key 1)
   TIME DELAY TO INPUT (SEC) == 0
   SYMBOL FRAC FOR SAMPLE TIME == 0
   SYMBOL RATE (HZ) == $CHIP RATE

GOLD PN SEQUENCE (key 2)
   USER NUMBER == $USER NUMBER
   SIZE == 1024

BUTTERWORTH LOWPASS (key 3)
   FILTER ORDER == 3
   PASSBAND EDGE (HZ) == 0.5 * 'CHIP RATE'
   PASSBAND EDGE ATTENUATION (DB) == 3

BUTTERWORTH LOWPASS (key 4)
   FILTER ORDER == 3
   PASSBAND EDGE (HZ) == 0.5 * 'CHIP RATE'
   PASSBAND EDGE ATTENUATION (DB) == 3

INTERFERING NOISE (UNCODED) (key 5)
   CHIP RATE == $CHIP RATE
   CHANNEL NUMBER == 25
```

```
    CHANNEL START NUMBER == 1
    CHANNEL VECTOR LENGTH == $CHANNEL VECTOR LENGTH
    DATA SEED == 183909825
    DELAY SEED == 699140313
    ACTIVATION SEED == 981312769
    UPDATE PERIOD == $UPDATE PERIOD

INTERFERING NOISE (UNCODED) (key 6)
    CHIP RATE == $CHIP RATE
    CHANNEL NUMBER == 8
    CHANNEL START NUMBER == 1
    CHANNEL VECTOR LENGTH == $CHANNEL VECTOR LENGTH
    DATA SEED == 1124259969
    DELAY SEED == 882912129
    ACTIVATION SEED == 742083457
    UPDATE PERIOD == $UPDATE PERIOD

INTERFERING NOISE (UNCODED) (key 7)
    CHIP RATE == $CHIP RATE
    CHANNEL NUMBER == 12
    CHANNEL START NUMBER == 1
    CHANNEL VECTOR LENGTH == $CHANNEL VECTOR LENGTH
    DATA SEED == 160204001
    DELAY SEED == 222021027
    ACTIVATION SEED == 310014729
    UPDATE PERIOD == $UPDATE PERIOD

INTERFERING NOISE (UNCODED) (key 8)
    CHIP RATE == $CHIP RATE
    CHANNEL NUMBER == 26
    CHANNEL START NUMBER == 1
    CHANNEL VECTOR LENGTH == $CHANNEL VECTOR LENGTH
    DATA SEED == 206419483
    DELAY SEED == 949161473
    ACTIVATION SEED == 1588745985
    UPDATE PERIOD == $UPDATE PERIOD

INTERFERING NOISE (UNCODED) (key 9)
    CHIP RATE == $CHIP RATE
    CHANNEL NUMBER == 45
    CHANNEL START NUMBER == 1
    CHANNEL VECTOR LENGTH == $CHANNEL VECTOR LENGTH
    DATA SEED == 1763284865
    DELAY SEED == 1404081793
    ACTIVATION SEED == 223961473
    UPDATE PERIOD == $UPDATE PERIOD

INTERFERING NOISE (UNCODED) (key A)
    CHIP RATE == $CHIP RATE
    CHANNEL NUMBER == 16
    CHANNEL START NUMBER == 1
    CHANNEL VECTOR LENGTH == $CHANNEL VECTOR LENGTH
    DATA SEED == 99142629
    DELAY SEED == 819856769
    ACTIVATION SEED == 1714069761
    UPDATE PERIOD == $UPDATE PERIOD
```

```
INTERFERING NOISE (UNCODED) (key B)
   CHIP RATE == $CHIP RATE
   CHANNEL NUMBER == 43
   CHANNEL START NUMBER == 1
   CHANNEL VECTOR LENGTH == $CHANNEL VECTOR LENGTH
   DATA SEED == 2027428865
   DELAY SEED == 1946191745
   ACTIVATION SEED == 1875028353
   UPDATE PERIOD == $UPDATE PERIOD

INTERFERING NOISE (UNCODED) (key C)
   CHIP RATE == $CHIP RATE
   CHANNEL NUMBER == 29
   CHANNEL START NUMBER == 1
   CHANNEL VECTOR LENGTH == $CHANNEL VECTOR LENGTH
   DATA SEED == 339568129
   DELAY SEED == 251663105
   ACTIVATION SEED == 387172481
   UPDATE PERIOD == $UPDATE PERIOD

INTERFERING NOISE (UNCODED) (key D)
   CHIP RATE == $CHIP RATE
   CHANNEL NUMBER == 32
   CHANNEL START NUMBER == 1
   CHANNEL VECTOR LENGTH == $CHANNEL VECTOR LENGTH
   DATA SEED == 2072676737
   DELAY SEED == 635605761
   ACTIVATION SEED == 1991407201
   UPDATE PERIOD == $UPDATE PERIOD

T_>= T_ON (key E)
   T-ON == $T-ON

PRINT SIGNAL (key F)
   LEADING STRING ==
   TRAILING STRING ==

INITIALIZATION CODE:
   (none)
```

Appendix 2

Fortran and C Support Code

A2.i Gold Sequence Generation

The Fortran code for generating all the true Gold sequences of length 2^k-1 from preferred pairs of maximal length sequences (m-sequences) of order k is given as PROGRAM_GOLD_GENERATOR. The method outlined in Chapter 3 and Appendix 7 of [15] and the tables of irreducible polynomials from [16] are used to generate the true Gold sequences. Since true Gold sequences of order 10 are desired, we find that the preferred pairs of m-sequences have characteristic polynomials of $1+X^3+X^{10}$ and $1+X^2+X^3+X^8+X^{10}$.

The program uses these characteristic polynomials to generate the two m-sequences and then combines them to generate the 2^k-1 true Gold sequences. These Gold sequences are then written to the file 'gold_codes.dat' which is to be read during the simulations. During the simulations, these true Gold sequences are modified by adding 1 chip of random state in order to make their lengths equal to 1024 (see Subsection 2.1.6).

```
C                PROGRAM_GOLD_GENERATOR

C      FORTRAN code for generating GOLD codes from two m-sequences
C      The length (n for code lengths of 2^n - 1), the number of
C      taps for each of the two m-sequences, and the tap numbers
C      of the two m-sequences must be altered within the program
C      to get different GOLD codes.  NOTE:  a maximum of 128 different
C      GOLD codes are generated.

       integer c, cc, ccc, a(5000), b(5000), out, n, golda(5000), max
       integer goldb(5000), backa, backb, fba(20), fbb(20), an, bn
       integer cona, conb


C      Make the two m-sequences first
C      n = the order of the m-sequences
C      an = the number of taps for the first m-sequence
C      bn = the number of taps for the second m-sequence
C      fba(x) and fbb(x) contain the values of the taps for the
C         first and second m-sequences respectively.

       n = 4
       an = 2
       bn = 2
       fba(1) = 1
       fba(2) = 4
       fba(3) = 3
       fba(4) = 4
       fba(5) = 999
       fba(6) = 999
       fbb(1) = 1
       fbb(2) = 2
       fbb(3) = 999
       fbb(4) = 999
       fbb(5) = 999
       fbb(6) = 999

C      Initialize the shift registers to all 1's.
       do 10 c = 1,n,1
       a(c) = 1
       b(c) = 1
    10 continue

C      Generate the two m-sequence codes.
       do 20 c = 1, 2**n-1, 1
       golda(c) = a(n)
       goldb(c) = b(n)

       backa = a(1)
       backb = b(1)

       cona = 0
       conb = 0

       do 30 cc = 1,an,1
        j = fba(cc)
        cona = cona + a(j)
    30 continue

       do 40 cc = 1,bn,1
```

```
          j = fbb(cc)
          conb = conb + b(j)
   40 continue

      if (cona/2 - cona/2.0.gt.-0.4) then
        a(1) = 0
      else
        a(1) = 1
      endif

      if (conb/2 - conb/2.0.ge.-0.4) then
        b(1) = 0
      else
        b(1) = 1
      endif

      do 50 cc = n,3,-1
        a(cc) = a(cc-1)
        b(cc) = b(cc-1)
   50 continue

      a(2) = backa
      b(2) = backb

   20 continue


C     Generate the 2**n-1 different GOLD codes by phase shifted
C     addition of the two m-sequence codes.

      open(unit=1,file='gold_codes.dat',status='unknown')
      write(1,1002) 2**n-1
 1002 format('',i5)
      write(1,1003)
 1003 format('')

      if (2**n-1.gt.128) then
          max = 128
      else
          max = 2**n-1
      endif
      do 100 c = 1,max,1
      do 110 cc = 1,2**n-1,1
        ccc = cc+c-1
        if (ccc.gt.2**n-1) then
           ccc = cc+c-2**n
        endif
        out = golda(cc) + goldb(ccc)
        if (out.ne.1) then
           out = -1
        endif
        write(1,1000) out
 1000 format(i2)
  110 continue
       write(1,1001)
 1001 format(' ')
  100 continue
      end
```

A2.ii Mean Cross-Correlation Between Partial and Complete Gold Sequence Determination

The code PROGRAM_CROSS_CORRELATION_FINDER is used to find the mean, maximum, and minimum cross-correlations of the partial and complete modified Gold sequences. The program accounts for even and odd cross-correlation values, all 1023 possible time shifts of one chip time between each pair of sequences, and smaller time shifts between sequences of 0.2 chip time.

The length of the partial sequences to compare and the number of complete sequences to use in the test can be specified by the user. The cross-correlations are given as magnitudes since the mean cross-correlation is used to determine the variance of the interference and cannot be negative.

```
C                PROGRAM_CROSS_CORRELATION_FINDER

C     FORTRAN sequence for getting mean partial and whole cross correlations
C        between Gold sequences generated by the gold_gen program.
C     This program includes possible
C     The maximum and minimum values are given as the output along
C        with the mean partial cross correlation.

      implicit none
      integer c,cc,ccc,co,cco,code(130,1026),rounds,max,max2
      integer plength,seqlen,numcode,s,ccccc,ph,seed,uni,brk1
      integer refstrt,jump,b,brk2,maxrounds
      real maxxcor,maxautocor,sum,tot,minxcor,mean,tk,otk,num
      real ran
      logical*1 swt,all

C     maxxcor is the maximum cross-correlation
C     maxautocor is the maximum auto-correlation
C     minxcor is the minimum cross-correlation
C     sum is used to find the correlation between current samples
C     code(x,y) is used to store the input gold codes:  up to 'x'
C          different codes of length 'y'
C     c, cc, ccc, ccccc, co, cco, s are counter variables
C     ph is a counter variable which determines the misalignment of
C       the gold sequence chips
C     tk and otk are the misalignments of the gold sequence chips.
C     max contains the value of the gold sequence's length
C     max2 contains the length of the extended gold sequences (= max+1)
C     rounds contains the value of the number of gold sequences to test
C     maxrounds contains the total number of gold sequences available
C       with a maximum value of 128
C     seqlen is the length of the sequences to partial correlate
C     plength is the number of partial sequence there are in the
C       whole gold sequence
C     tot sums all obtained values of the variable 'sum'
C     num keeps track of the number of values of 'sum' obtained
C     mean = tot/num which gives the mean xcorrelation of the partial sequences
C     numcode is the number of whole gold sequences to include in this test
C     seed holds the seed for the random number generator which determines
C       which codes are used in the cross-correlation test
C     refstrt is the randomly chosen first code used in the test
C     jump is a randomly determined variable which determines which
C       codes are used in the test
C     uni is a function which produces randomly generated +1s and -1s
C     brk1, brk2 are the states of the data bit which are being spread
C       by the Gold sequences
C     b is a counter variable which is used to run the program once for
C       similar adjacent data bits and once for dissimilar adjacent data bits
C     ran is the random number generator function
C     swt is a logical variable which signals when the data bit has changed
C       state
C     all is .true. if all sequences are to be tested

      seqlen = 128
      numcode = 3
      all = .false.


C     Read the Gold sequence length and the determine the number
C        of codes to test.
```

```
      open(unit=1,file='gold_codes.dat',status='old')
      read(1,*) max
      if (max.gt.numcode) then
         rounds = numcode
      else
         rounds = max-1
      endif
      maxrounds = max
      if (maxrounds.gt.128) then
        maxrounds = 128
      endif

C     Read seed for random number generator which is to be used
C      for extending Gold sequences and selecting which codes
C      to test.
      open(unit=2,file='seed.dat',status='old')
      read(2,*) seed
      close(2)

C     Initialize parameters.
      maxxcor = -seqlen
      maxautocor = seqlen
      minxcor = seqlen
      sum = 0.0
      num = 0.0
      plength = (max+1)/seqlen
      swt = .false.
      refstrt = int((maxrounds-rounds)*ran(seed))
      jump = int((maxrounds-refstrt)*ran(seed)/rounds)
      if  (jump.lt.1) then
        jump = 1
      endif
      if (all) then
        refstrt = 1
        jump = 1
        rounds = max-1
      endif
      write(*,*) 'refstart =',refstrt,'  jump =',jump

C     Write new seed to file.
      open(unit=3,file='seed.dat',status='old')
      write(3,*) seed
      close(3)

C     Read the gold sequences from the file.
      max2 = max+1
      do c = 1,maxrounds,1
        do cc = 1,max,1
           read(1,*) code(c,cc)
        enddo
        code(c,max2) = uni(seed)
      enddo


C     Test cross correlations between all combinations
C      of codes and all phase shifted combinations of codes.
C     b is the counter for a change in state of the data being
C       spread by the test gold sequence.
C     ph determines the amount of misalignment between chips
C     co is the counter for the reference gold sequence.
```

```
C     cco is the counter for the test gold sequence.
C     c is the counter for the phase shift.
C     cc is the counter for the chips of the reference gold sequence.
C     ccc is the counter for the chips of the test gold sequence.
C     ccccc is the counter for the chips of the reference gold sequence.
C     s counts the number of partial sequences
      do b = 1,2,1
      do ph=1,10,2
        tk = (ph-1)*0.1
        otk = 1.0-tk
      do co = refstrt,refstrt+rounds*jump-1,jump
       do cco = co+jump,refstrt+rounds*jump,jump
        do c = 1,max2,1
         brk1 = 1
         brk2 = 1
         do s = 1,plength,1
          do cc = (s-1)*seqlen+1,s*seqlen,1
            ccc = cc+c-1
            ccccc = ccc+1
            if (ccccc.gt.max2) then
              ccccc = ccccc-max2
            endif
            if (ccc.gt.max2) then
              ccc = ccc-max2
            endif
            if (b.eq.2) then
              if (swt) then
                 brk1 = brk2
                 swt = .false.
              else
                if (mod(ccccc,seqlen)-1.eq.0) then
                  brk2 = brk2-2*brk2
                  swt = .true.
                endif
              endif
            endif
            sum=sum+code(co,cc)*(otk*code(cco,ccc)*brk1 +
     &                           tk*code(cco,ccccc)*brk2)
          enddo
           sum = abs(sum)
           tot = tot + sum
           num = num + 1.0
           if (sum.gt.maxxcor) then
             maxxcor = sum
           endif
           if (sum.lt.minxcor) then
              minxcor = sum
           endif
           sum = 0.0
         enddo
        enddo
       enddo
      write(*,*) 'finished sequence',b,ph,co
      enddo
      enddo
      enddo
      mean = tot/num

C     Display maximum and minimum values of auto and cross correlations.
      write(*,*) '','maxautocorr = ',maxautocor
```

```
      write(*,1005) maxxcor
1005  format(' ','maxxcorr = ',f8.3)
      write(*,1010) minxcor
1010  format(' ','minxcorr = ',f8.3)
      write(*,*) 'Mean Xcorrelation = ',mean
      write(*,*) 'tot =',tot,'  num =',num
      end

      integer function uni(seed)
        integer seed
        real ran,x
        x = ran(seed)
        if (x.gt.0.5) then
          uni = 1
        else
          uni = -1
        endif
      return
      end
```

A2.iii Valid Orthogonal Convolutional Codeword Generation

The C code used to generate all the valid orthogonal convolutional codewords for a code of constraint length K (for K > 2) is given as PROGRAM_OC_GEN. The codeword generation is modelled after the generator given in Figure 6.

```
/*                      PROGRAM_OC_GEN
*
*
*     Generator for all Orthogonal Convolutional Codes for
*                      a Given Value of K
*
*
*/

#include <stdio.h>
#include <math.h>

/*  Global Variables  */
int a[500000], out[20][500000], k, run, c1, c2, c3, c4, t, flag[20];
int c6, c7;
double c5,1;

/*****************************************************************************/
stage1()
{
  if ( fabs( (double)(t/2.0 - t/2)) < 0.0001)
    {
    out[1][t] = a[1];
    }
  else
    {
     out[1][t] = 0;
    }
}

/*****************************************************************************/
stage2()
{
  if (fabs( (double)((t-c2)/2.0 - (t-c2)/2) ) < 0.0001)
    {
    flag[2] = abs(flag[2]-1);
    }
  if (flag[2] == 1)
    {
    out[2][t] = out[1][t-2] + a[2];
    if (out[2][t] == 2)
      {
      out[2][t] = 0;
      }
    }
  else
    {
    out[2][t] = out[1][t];
    }
}


/*****************************************************************************/
stagebeyond()
{
  for (c3=3; c3 <=k; c3++)
    {
    c4 = (int)(pow(2.0,(double)(c3-1)));
    c5 = (double)(c4);
    if (fabs( (double)((t-c2)/c5 - (t-c2)/c4)) < 0.0001)
```

```
        {
        flag[c3] = abs(flag[c3]-1);
        }
      if (flag[c3] == 1)
        {
        out[c3][t] = out[c3-1][t-c4] + a[c3];
        if (out[c3][t] == 2)
          {
          out[c3][t] = 0;
          }

        }
      else
        {
        out[c3][t] = out[c3-1][t];

        }
      }
}

/*************************************************************************/
main(argc,arg1)
int argc;
char *arg1[];
{
  FILE *outfile;

  int atoi();
  if (argc != 2)
    {
    printf("\n\nERROR:  format must be 'gen_all_conv_codes.exe k'\n\n");
    exit(1);
    }
  else
    {
    k = atoi(arg1[1]);
    run = (int)(pow(2.0,(double)(k)));
    }

  outfile = fopen("all_codes.dat","w");
  fprintf(outfile,"%d\n%d\n",k,run);

  for (c1=1; c1<=k; c1+=1)
    {
    a[c1] = 0;
    flag[c1] = 1;
    }
  c6 = -1;

  for (c2=1; c2<=(run*(int)(pow(2.,(double)(k)))); c2+=(int)(pow(2.,(double)(k))))
    {
    c6++;
      for (c7=0; c7<=(k-1); c7++)
        {
        if (((1<<c7) & c6) != 0)
          {
          a[c7+1] = 1;
          }
        else
          {
```

```
            a[c7+1] = 0;
            }
          fprintf(outfile,"%d",a[c7+1]);
          }
      fprintf(outfile,"\n");

      for (t=c2; t<=(c2+(int)(pow(2.,(double)(k)))-1); t++)
        {
        stage1();
        stage2();
        stagebeyond();
        if (out[k][t] == 1)
           out[k][t] = -1;
        if (out[k][t] == 0)
          out[k][t] = 1;
        fprintf(outfile,"%d\n",out[k][t]);
      }

/*    fprintf(outfile,"\n")*/;
    }
  fclose(outfile);
}
```

A2.iv Statistical Analyses (Mean and Variance Measurement)

The program PROGRAM_STAT_ANALYSIS finds the mean and variance of a set of complex data points in the file 'stat1.in'. The output, containing the mean and the variance of the data set, the minimum value, the maximum value, the number of (0.0,0.0) points, and the total number of data points in the data set is written to the file 'stat.dat'. The cumulative distribution is written to the file 'cdf.dat'.

```
C                    PROGRAM_STAT_ANALYSIS

C     This program performs statistical analysis on a set of
C     complex data points in the file 'stat1.in'.  The data must
C     be in the form '(##, ##)' representing a complex number.
C
C     The means and variances of the real and imaginary parts of
C     the data are determined first.  A pdf is generated.  This
C     pdf is different in that the data is binned and the prob-
C     ability of data in each bin is equal.  The bin size is varied
C     to make this so.  The file 'stat.out' contains this data.
C
C     Some possibly useful analysis information is written to the
C     file 'stat.dat'.
C
C     This program also generates the file 'cdf.dat' which holds
C     the cdf of the input data.


      implicit none
      complex in,csum,cmean,cmplx
      real total,rsumsq,rsum,rmean,rnum(1000000)
      real rvar,rmin,rmax,rspc,numbin,pi
      integer i,k,step,numvar,totzero,totali
      integer spc
      logical*1 elim_zero,detect_zero,cmpx


      open(unit=1,file='stat1.in')
      open(unit=3,file='stat.dat')
      open(unit=4,file='cdf.dat')

C     The value 'numbin' determines how many bins to divide
C      the input data into when determining probabilities.
C     The value 'numvar' determines how many different
C      variances for the gaussian distribution are tested.
C      These variances are set at multiples of the 'measured
C      test variance/numvar' and end when the variance reaches
C      the 'measured test variance'.
C     If gauss_go is .false., then no comparison will be made
C      between the input data and a gaussian distribution.
C     If elim_zero is .true., then any data points which equal
C      (0.,0.) are omitted from the statistical analysis.  The total
C      number of zeros is summed.
C     If cmpx is .true., then the data is treated as complex numbers.
C      If it is .false., then the data is treated as two sets of
C      real numbers.
      numbin = 120.0
      step = 4
      detect_zero = .true.
      cmpx = .false.
      elim_zero = .true.
      numvar = 1


C     Find the maximum and minimum components of the real part
C     of the input complex numbers.  Collect sums for the real
C     components of the input complex numbers in order to de-
C     termine the real mean.
      total = 0.0
```

```
      k = -1
      pi = 4.0*atan(1.0)
      totali = 0
    1 read(1,*,end=100) in
      if (detect_zero.and.(in.eq.(0.0,0.0))) then
        totzero = totzero + 2
        if (elim_zero) then
          goto 1
        endif
      endif
      totali = totali + 2
      k = k+1
      if (mod(k,step).eq.0) then
        if (.not.cmpx) then
          rsum = rsum + real(in)+imag(in)
        else
         csum = csum + in
        endif
        rnum(totali) = real(in)
        rnum(totali+1) = imag(in)
        if (real(in).gt.rmax.or.imag(in).gt.rmax) then
          rmax = max(real(in),imag(in))
        else
          if (real(in).lt.rmin.or.imag(in).lt.rmin) then
            rmin = min(real(in),imag(in))
          endif
        endif
        total = total+2.0
      endif
      goto 1
  100 continue
      close(1)


C     Calculate the average mean of the real and imaginary parts of
C     the input complex number.
      if (cmpx) then
        cmean = 2.0*csum/total
      else
        rmean = rsum/total
      endif

C     Get sums in order to compute the AVERAGE variance of the real
C     AND imaginary parts or the variance of the complex numbers.
      if (.not.cmpx) then
        rsumsq = 0.0
        do i = 2,totali,1
          rsumsq = rsumsq + (rnum(i)-rmean)**2.0
        enddo
        rvar = rsumsq/(total-1.0)
      else
        rsumsq = 0.0
        do i = 2,totali,2
          rsumsq = rsumsq + cabs(cmplx(rnum(i),rnum(i+1)))**2
        enddo
        rvar = rsumsq/(total/2.0-1.0)-cabs(cmean)**2.0
      endif

      call sort(totali,rnum)
```

```
C     Write the cdf file into 'cdf.dat'.  This part works when the data
C     is treated as two sets of real numbers only!!!
      spc = totali/numbin
      rspc = 1.0/numbin
      write(4,*) rnum(1),0.0
      do i = spc,totali,spc
        write(4,*) rnum(i),i/total
      enddo
      if (mod(totali,spc).ne.0) then
        write(4,*) rnum(totali),1.0
      endif


C     Write some useful data information.
      if (cmpx) then
        write(3,*) 'mean = ',cmean,'   variance = ',rvar
      else
        write(3,*) 'mean = ',rmean,'   variance = ',rvar
      endif
      write(3,*) 'min = ',rmin,'  max = ',rmax
C      write(3,*) 'resolution = 1/numbins = ',1.0/numbin
      if (cmpx) then
        write(3,*) 'total number of non-zero points = ',int(total/2)
        write(3,*) 'total number of zero points = ',totzero/2
      else
        write(3,*) 'total number of non-zero points = ',int(total)
        write(3,*) 'total number of zero points = ',totzero
      endif

      close(3)

      close(2)

 1000 format('mean = ',e16.8,'  var = ',e16.8)

      end


      subroutine sort(n,ra)
      real ra(n),rra
      integer l,ir,i,j
      l = n/2+1
      ir = n
   10 continue
      if (l.gt.1) then
        l = l-1
        rra = ra(l)
      else
        rra = ra(ir)
        ra(ir) = ra(1)
        ir = ir-1
        if (ir.eq.1) then
          ra(1) = rra
          return
        endif
      endif
      i = l
      j = l+1
   20 if (j.le.ir) then
        if (j.lt.ir) then
```

```
      if (ra(j).lt.ra(j+1)) then
        j = j+1
      endif
    endif
    if (rra.lt.ra(j)) then
      ra(i) = ra(j)
      i = j
      j = j+j
    else
      j = ir+1
    endif
  goto 20
  endif
  ra(i) = rra
  goto 10
  end
```

A2.v Kolmogorov-Smirnov Test

The code PROGRAM_KS_TEST performs the one-sided Kolmogorov-Smirnov test for the data set in 'stat1.in' and a Gaussian distribution. The mean and variance of the Gaussian distribution specified in the program should be the same as that of the test data. They can be found by using PROGRAM_STAT_ANALYSIS. The result of this test is given in units of level of significance. The closer this value is to one, the more alike the data set is like the Gaussian distribution.

```
C                          PROGRAM KS_TEST

C     This program performs the Kolmogorov-Smirnov Test on one binned or
C     continuous data set and a theoretical gaussian distribution variable
C     variance and mean.  The result is computed is the level of significance
C     and is closer to 1 if the data sets are closely related.  A value
C     greater than 0.05 is usually acceptable to not reject the null
C     hypothesis that the two are similar.

      real dat1r(500000)

      complex in
      integer i,k,numset1,reads,jump1,l,numbin
      real var,mean,prob,d
      logical*1 elim_zero,bin

C     The value of 'reads' determines the maximum number of data
C      points to be used in the test.  The numbers contained in
C     the 'jump1' variables represent the 'mod' number to use
C      when determining which input variables to actually use.
C     The logical variable 'elim_zero' determines whether or not
C      to pass over any inputs which equal (0.0,0.0).  A .true. value
C      will cause the program to skip all (0.0,0.0) inputs, a .false.
C      value will not.
C     The logical variable 'bin' determines whether to treat the data
C      set as continuous or binned data.
C     The variable 'numbin' indicates how many bins to divide the data
C      into (used only if 'bin' = .true.).
      reads = 1000000
      jump1 = 4
      var = 1.47038
      mean = 0.0
      elim_zero = .true.
      bin = .false.
      numbin = 100
      open(unit=1,file='stat1.in')


C     Read and store the input data.
      i = 0
      k = 0
    1 read(1,*,end=100) in
      if (elim_zero.and.(in.eq.(0.0,0.0))) then
        goto 1
      endif
      if (mod(l,jump1).eq.0) then
        dat1r(i+1) = real(in)
        dat1r(i+2) = imag(in)
        i = i+2
        if (i/2.eq.reads) then
          goto 100
        endif
      endif
      l = l + 1
      goto 1
  100 l = 0
  101 continue

  200 numset1 = i
      close(1)
```

```
C     Call the ksone routine to perform the ks test.
      call ksone(dat1r,numset1,d,prob,numbin,bin,var,mean)

C     Write the value of 'd' which is the maximum difference
C     between the cumulative distributions, and the value of
C     prob, which is the significance of the test.
C     A small value of 'd' and a 'prob' value close to 1.0
C     indicate that the two data sets are likely from the
C     same distribution.
      write(*,*) 'number of data points = ',numset1
      write(*,*) 'd = ',d,'  prob = ',prob

      end



C     **** The following routines were modified from: ****
C     Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vettering, W.T.,
C       'Numerical Recipes:  the art of scientific computing',
C        Cambridge University Press, 1986.

C     The following subroutine performs the Kolmogorov-Smirnov test.
      subroutine ksone(dat1,n1,d,prob,numbin,bin,var,mean)
      implicit none
      real dat1(*),fo,d,prob,dt,fn,fnc,en,ff,var,mean,probks
      integer n1,j,spc,numbin
      logical*1 bin

      call sort(n1,dat1)
      en = n1
      fo = 0.0
      d = 0.0
      if (.not.bin) then
      do j = 1,n1
        fn = j/en
        ff = fnc(dat1(j),var,mean)
        dt = max(abs(fo-ff),abs(fn-ff))
        if (dt.gt.d) then
           d = dt
        endif
        fo = fn
      enddo
      prob = probks(sqrt(en)*d)
      return

      else

      spc = n1/numbin
      do j = 1,n1,spc
        ff = fnc(dat1(j),var,mean)
        fn = j/en
        dt = abs(ff-fn)
        if (dt.gt.d) then
          d = dt
        endif
      enddo
      prob = probks(sqrt(float(numbin))*d)
      return
      endif
```

```
      end


C     This function calculates the significance of the test.
      function probks(alam)
      implicit none
      real alam,a2,fac,probks,termbf,term
      integer j
      a2 = -2.0*alam**2
      fac=2.0
      probks = 0.0
      termbf = 0.0
      do j = 1,100,1
        term = fac*exp(a2*j**2)
        probks = probks+term
        if (abs(term).lt.0.001*termbf) then
          return
        endif
        fac = -fac
        termbf = abs(term)
      enddo
      if (probks.lt.1E-3) then
         probks = 0.0
      else
        probks = 1.0
      endif
      return
      end


C     This routine sorts the arrays into increasing order.
      subroutine sort(n,ra)
      implicit none
      real ra(*),rra
      integer l,ir,i,j,n
      l = n/2+1
      ir = n
   10 continue
      if (l.gt.1) then
        l = l-1
        rra = ra(l)
      else
        rra = ra(ir)
        ra(ir) = ra(1)
        ir = ir-1
        if (ir.eq.1) then
          ra(1) = rra
          return
        endif
      endif
      i = l
      j = l+1
   20 if (j.le.ir) then
        if (j.lt.ir) then
          if (ra(j).lt.ra(j+1)) then
            j = j+1
          endif
        endif
        if (rra.lt.ra(j)) then
```

```
          ra(i) = ra(j)
          i = j
          j = j+j
        else
          j = ir+1
        endif
      goto 20
      endif
      ra(i) = rra
      goto 10
      end


C     This function calculates the cdf of a gaussian distribution at value x.
      function fnc(x,var,mean)
      real x,var,mean

C      fnc=1.0-0.5*erfc((x-mean)/sqrt(2.0*var))
      fnc = .5+.5*erf((x-mean)/sqrt(2.0*var))
      return
      end
```

A2.vi Channel Formatter

The program PROG_CHAN_FORM alters the data format of the channel profiles generated by SIRCIM. The magnitude and phase components of the SIRCIM channel impulse responses are converted into the real and imaginary components. Also, the energy seen by the receiver when a square pulse of duration equal to one chip time propagates through the channel is normalized to unity.

```
C                          PROG_CHAN_FORM

C     This program is used to equalize the energy of the impulse power
C     profile contained in the file 'chnl.dat'.  The energy contained in
C     the time domain impulse response when it is time convolved with a
C     square pulse of duration = 4*7.8125 ns is normalized to 1.0.
C     This program will then convert the power profiles into normalized
C     real and imaginary components of the impulse response.
C     Note that this output is in the form of absolute signal strength,
C     not power with respect to the value 10**(-3.8) as generated by
C     SIRCIM.
C     The converted data will be written to the file 'chnl.norm'.

      implicit none
      integer i,j,num,len,k,samp_sym
      real h(65),ph(65),dist(65),time(65),sum,pi
      complex h1(65),cmplx,out(80)

C     num = 19 for channels uninterpolated by cspline.exe
C     num = 181 for channels interpolated by cspline.exe

      num = 19

      open(unit=1,file='chnl.dat',status='unknown')
      open(unit=2,file='chnl.norm',status='unknown')
      pi = 4.0*atan(1.0)
      samp_sym = 4
      len = 65
      sum = 0.0

C     Read the channel profiles.
      do j = 1,num,1
      sum = 0.0
      do i = 1,len,1
      read(1,*) dist(i),time(i),h(i),ph(i)
      enddo

C     Change into rectangular coordinates.
      do i = 1,len,1
      h(i) = sqrt(h(i))
      h1(i) = h(i)*cmplx(cos(ph(i)*pi/180.0),sin(ph(i)*pi/180.0))
      enddo

C     Find output when channel is time convolved with a square pulse
C     of duration equal to the time between 'samp_sym' impulses in the
C     channel profile.
      do i = 1,len+samp_sym-1,1
        do k = 1,samp_sym,1
          if (i-k+1.ge.1) then
            if (i-k+1.le.len) then
              out(i) = out(i) + h1(i-k+1)
            endif
          endif
        enddo
      enddo

C     Find factor used to normalize the channel.
      do i = 1,len+samp_sym-1,1
       sum = sum + cabs(out(i))**2.0
```

```
      enddo
      write(*,*) sum

C     Normalize the channel.
      do i = 1,len,1
        h1(i) = h1(i)/sqrt(sum)
      enddo

C     Write the normalized channel file.
      do i = 1,len,1
      write(2,100) dist(i),time(i),h1(i)
      enddo

  100 format(f10.6,f12.6,e14.6,e14.6)

      do i = 1,len+samp_sym-1,1
        out(i) = cmplx(0.0,0.0)
      enddo

      enddo
      close(1,status='keep')
      close(2,status='keep')
      end
```

A2.vii Error Bound Calculation Program

The program PROG_OC_BER calculates the value of the error bound given by equation (2.2.3.11). The user must enter the value of 'pe', which corresponds to the value of p in the equation, and the code's constraint length into the program. The program will then calculate the value of error bound value P_b.

```
C                          PROG_OC_BER

C          Orthogonal Convolutional BER from Symbol Error Calculator

C     This program finds the bit error rate from the symbol error rate
C     for an orthogonal convolutional code of user defined constraint
C     length in a binary symmetric channel.

      implicit none
      integer n,constraint
      real*8 Z,pe,pdata

C     User Defined Parameters
      constraint = 3
      pe = 250./25755
      write(*,*) ''
      write(*,*) 'pe =',pe


C     Variable Initialization
      n = 2**constraint

C     BER Calculation
      Z = (4.0*pe*(1-pe))**0.5
      pdata = Z**(constraint*n/2.0)/(1-2.0*Z**(n/2.0))**2.0

      write(*,*) 'Pb =',pdata
      write(*,*) ''
      end
```

A2.viii Channel Analysis Program

The program PROG_CHANNEL_ANAL performs the channel analysis method on any given channel profile. The channel file 'fc*xx*a' (where *xx* is a user specified number from 1 to 50 inclusive) is read and the appropriate profile within the file is found. The normalized mean cross-correlation between the SSMA spreading codes must be specified in order to determine the proper signal to interference ratio.

The program finds the optimum sampling points within the channel and determines the signal to noise ratios for all L channels of the multichannel DPSK receiver (which is used to represent a L-stage RAKE receiver). The resulting symbol is calculated.

This program allows the user to specify whether or not orthogonal convolutional codes of any constraint length are to be used. If they are, then the resulting symbol error rate is substituted into the error bound for the code. If they are not, then the resulting symbol error rate is also the bit error rate.

```
C                    PROG_CHANNEL_ANAL

C     This program finds the appropriate delay required in order
C     to sample the signal at the optimum time.  This delay accounts
C     for the multipath channel only.  The number of samples per
C     symbol time, the diversity order of the RAKE (equal gain)
C     receiver, the channel length (in # of samples), the channel
C     set to use, and the impulse response within the set to be used
C     need to be specified.  The units of the output 'delay' is in
C     number of samples.

C     The program then finds the percentage of energy that the RAKE
C     receiver will use out of all the energy of the channel.

C     Using this value, the symbol error probability is calculated.
C     In this calculation, the multipath diversity order, the cross-
C     correlation between pn sequences, and the number of interfer-
C     ing SSMA users is taken into account.  From this symbol error
C     probability, the bit error probability is calculated from
C     the code rate parameter.

C     If the variable 'coded' is .false., then the symbol error
C     probability is equal to the average bit error rate of an
C     uncoded system.

      implicit none
      integer lun,nol,chlstrt,chnlnum,i,j,k,delay
      integer len,diver,samp_sym,count,next,users,L,n,constraint
      complex ch(80),sum(80),cmplx,out(80)
      real dist,time,mag,max,pmag(10),ppmag(10),sig3(10)
      real pb,xcorr,Z,pdata,chpwr,pemin,sig(80),sig2(10)
      logical*1 coded

C     User Defined Parameters
      chnlnum = 39
      chlstrt = 12
      coded = .false.
      constraint = 5
      users = 9
      xcorr = 19.7768/1024.0
      L = 3


C     Variable Initialization
      nol = 1
      lun = 2
      samp_sym = 4
      diver = 3
      len = 65
      if (coded) then
        n = 2**constraint
      endif
      pemin = 1.0


C     Read the appropriate channel impulse response
        go to(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,
     &        21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,
```

```
     &        37,38,39,40,41,42,43,44,45,46,47,48,49,50),chnlnum
     1   open(unit=no1,file='fc1a',status='old')
         go to 99
     2   open(unit=no1,file='fc2a',status='old')
         go to 99
     3   open(unit=no1,file='fc3a',status='old')
         go to 99
     4   open(unit=no1,file='fc4a',status='old')
         go to 99
     5   open(unit=no1,file='fc5a',status='old')
         go to 99
     6   open(unit=no1,file='fc6a',status='old')
         go to 99
     7   open(unit=no1,file='fc7a',status='old')
         go to 99
     8   open(unit=no1,file='fc8a',status='old')
         go to 99
     9   open(unit=no1,file='fc9a',status='old')
         go to 99
    10   open(unit=no1,file='fc10a',status='old')
         go to 99
    11   open(unit=no1,file='fc11a',status='old')
         go to 99
    12   open(unit=no1,file='fc12a',status='old')
         go to 99
    13   open(unit=no1,file='fc13a',status='old')
         go to 99
    14   open(unit=no1,file='fc14a',status='old')
         go to 99
    15   open(unit=no1,file='fc15a',status='old')
         go to 99
    16   open(unit=no1,file='fc16a',status='old')
         go to 99
    17   open(unit=no1,file='fc17a',status='old')
         go to 99
    18   open(unit=no1,file='fc18a',status='old')
         go to 99
    19   open(unit=no1,file='fc19a',status='old')
         go to 99
    20   open(unit=no1,file='fc20a',status='old')
         go to 99
    21   open(unit=no1,file='fc21a',status='old')
         go to 99
    22   open(unit=no1,file='fc22a',status='old')
         go to 99
    23   open(unit=no1,file='fc23a',status='old')
         go to 99
    24   open(unit=no1,file='fc24a',status='old')
         go to 99
    25   open(unit=no1,file='fc25a',status='old')
         go to 99
    26   open(unit=no1,file='fc26a',status='old')
         go to 99
    27   open(unit=no1,file='fc27a',status='old')
         go to 99
    28   open(unit=no1,file='fc28a',status='old')
         go to 99
    29   open(unit=no1,file='fc29a',status='old')
         go to 99
    30   open(unit=no1,file='fc30a',status='old')
```

```
      go to 99
 31   open(unit=no1,file='fc31a',status='old')
      go to 99
 32   open(unit=no1,file='fc32a',status='old')
      go to 99
 33   open(unit=no1,file='fc33a',status='old')
      go to 99
 34   open(unit=no1,file='fc34a',status='old')
      go to 99
 35   open(unit=no1,file='fc35a',status='old')
      go to 99
 36   open(unit=no1,file='fc36a',status='old')
      go to 99
 37   open(unit=no1,file='fc37a',status='old')
      go to 99
 38   open(unit=no1,file='fc38a',status='old')
      go to 99
 39   open(unit=no1,file='fc39a',status='old')
      go to 99
 40   open(unit=no1,file='fc40a',status='old')
      go to 99
 41   open(unit=no1,file='fc41a',status='old')
      go to 99
 42   open(unit=no1,file='fc42a',status='old')
      go to 99
 43   open(unit=no1,file='fc43a',status='old')
      go to 99
 44   open(unit=no1,file='fc44a',status='old')
      go to 99
 45   open(unit=no1,file='fc45a',status='old')
      go to 99
 46   open(unit=no1,file='fc46a',status='old')
      go to 99
 47   open(unit=no1,file='fc47a',status='old')
      go to 99
 48   open(unit=no1,file='fc48a',status='old')
      go to 99
 49   open(unit=no1,file='fc49a',status='old')
      go to 99
 50   open(unit=no1,file='fc50a',status='old')
 99   continue
      do i = 1,chlstrt,1
        do k = 1,len,1
          read(no1,100) dist,time,ch(k)
100       format(f10.6,f12.6,e14.6,e14.6)
        enddo
      enddo
      close(no1,status='keep')


C     Find the number of sample delays required so that sampling will
C     be done at the optimum point (maximum eye opening).  If diver-
C     sity is used, then this is also taken into account.  The total
C     energy used by the Lth order RAKE receiver and the total energy
C     of the channel is also calculated.

C     Find output of square pulse convolved with channel profile.
      do i = 1,len+samp_sym-1,1
        do j = 1,samp_sym,1
          if (i-j+1.ge.1) then
```

```
            if (i-j+1.le.len) then
              out(i) = out(i) + ch(i-j+1)
            endif
          endif
        enddo
      enddo

C     Find total power of one square pulse after channel propagation.
      do i = 1,len+samp_sym-1,1
        chpwr = chpwr + cabs(out(i))**2
      enddo

      do count = 1,len+samp_sym-1-samp_sym*diver,1
        do j = 1,diver,1
          pmag(j) = 0.0
          next = (j-1)*samp_sym+count
          do i = next,next+samp_sym-1,1
            sum(count) = sum(count)+out(i)
            sig(count) = sig(count)+cabs(out(i))**2.0
          enddo
          pmag(j) = cabs(sum(count))**2.0
          sig2(j)  = sig(count)
          mag = cabs(sum(count))**2.0+mag
          sum(count) = cmplx(0.0,0.0)
          sig(count) = cmplx(0.0,0.0)
        enddo

        if (mag.gt.max) then
          delay = count
          max = mag
          do j = 1,diver,1
            ppmag(j) = pmag(j)
            sig3(j) = sig2(j)
          enddo
        endif
        mag = 0.0
      enddo


      pemin = pb(users,xcorr,ppmag,diver,chpwr,L,sig3)


      write(*,*) 'Channel ',chnlnum,'-',chlstrt
      write(*,*) '  delay =',delay
      write(*,*) '  channel power = ',chpwr
      write(*,*) '  symbol error rate =',pemin
      if (coded) then
          Z = (4.0*pemin*(1-pemin))**0.5
          pdata = Z**(constraint*n/2.0)/(1-2.0*Z**(n/2.0))**2.0
      else
          pdata = pemin
      endif
      write(*,*) '  bit error rate =',pdata
      end


C     This function calculates the bit error probability of
C     a L-path DPSK receiver.
      real function pb(users,xcorr,pstage,diver,total,L,sig3)
```

```
      real xcorr,snr,c,sum1,total,Eint,Esint,pstage(*),sig3(*)
      integer L,users,n,n1,diver,i
      snr = 0.0
      do i = 1,diver,1
        Eint = (users*xcorr*total/2.0)
        Esint = (total - sig3(i))*xcorr
        snr = snr+.25*pstage(i)/(Eint+Esint)
        write(*,*) 'SNR(',i,') =',.25*pstage(i)/(Eint+Esint)
        write(*,*) '  pstage(',i,') =',pstage(i)
        write(*,*) '  Esint(',i,') =',Esint
        write(*,*) '  Eint(',i,') =',Eint
      enddo
      sum1 = 0.0
      do n1 = 1,L,1
        n = n1 - 1
        sum1 = snr**n * c(n,L) + sum1
      enddo
      pb = exp(-snr)*sum1/2**(2*L-1)
      return
      end


C     This function calculates the value of:
C
C       (1/n!) * SUM[ (2*L-1)!/(k!*(2*L-1-k)!) ]
C
C     where the summation is from k = 0 to L-1-n
      real function c(n,L)

      real b,sum2
      integer n,k,L,k1,fc
      c = 0.0
      sum2 = 0.0
      do k1 = 1,L-n,1
        k = k1 - 1
        sum2 = sum2+b(2*L-1,k)
      enddo
      c = (1/fc(n))*sum2
      return
      end

C     This function calculates the value of x!/(y!*(x-y)!)
      real function b(x,y)

      integer x,y,fc
      b = real(fc(x))/real(fc(y)*fc(x-y))
      return
      end


C     This function calculates x!
      integer function fc(x)

      integer x,a
      fc = 1
      do a = x,1,-1
        fc = fc*a
      enddo
      return
      end
```